

# LightWave™ 10



خالد الحقي

©Muharraql-Studios

## SURFACING AND RENDERING



# LightWave<sup>®</sup> 10

## Reference Manual

License.key number

Win32	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	—	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	—	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	—	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
Win64	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	—	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	—	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	—	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
MAC	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	—	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	—	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	—	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>

This publication, or parts thereof, may not be reproduced in any form, by any method, for any purpose without prior written consent from NewTek, Inc.

© Copyright 2010, NewTek, Inc. All Rights Reserved

LightWave 3D™

NewTek  
5131 Beckwith Blvd  
San Antonio, TX 78249  
Tel.: 1-800-862-7837 - Fax: 210-370-8001  
[www.newtek.com](http://www.newtek.com)





## LightWave 3D® Software License and Limited Warranty

PLEASE READ CAREFULLY BEFORE INSTALLING AND/OR USING THIS SOFTWARE. BY INSTALLING AND/OR USING THIS SOFTWARE, YOU AGREE TO BECOME BOUND BY THE TERMS OF THIS LICENSE. IF YOU DO NOT AGREE TO THE TERMS OF THIS LICENSE, RETURN THIS PACKAGE TO THE PLACE WHERE YOU OBTAINED IT WITHIN 15 DAYS OF PURCHASE FOR A FULL REFUND.

### 1. Grant of License

The enclosed computer program(s) (the "Software") is licensed, not sold, to you by NewTek, Inc. (NEWTEK) for use only under the terms of this License, and NEWTEK reserves any rights not expressly granted to you. You own the disk(s) on which the Software is recorded or fixed, but the Software and all copyright rights therein, foreign and domestic, is owned by NEWTEK or its suppliers and is protected by United States copyright laws and international treaty provisions.

The copyright restrictions of this license extend to any further updates, software patches, or bug fixes made available to you by NEWTEK, whether distributed by floppy disc, CD ROM, DVD ROM or in an electronic format via BBS, ftp, email, etc.

This License allows you to use one copy of the Software on a single computer at a time. To "use" the Software means that the Software is either loaded in the temporary memory (i.e., RAM) of a computer, or installed on the permanent memory of a computer (i.e., hard disk, CD ROM, DVD ROM, etc.).

You may use at one time as many copies of the Software as you have licenses for. You may install the Software on a common storage device shared by multiple computers, provided that if you have more computers having access to the common storage device than the number of licensed copies of the Software, you must have some software mechanism which locks out any concurrent user in excess of the number of licensed copies of the Software (an additional license is not needed for the one copy of Software stored on the common storage device accessed by multiple computers).

You may make one copy of the Software in machine readable form solely for backup purposes. The Software is protected by copyright law. As an express condition of this License, you must reproduce on the backup copy the NEWTEK copyright notice in the following format "(c) 1990 - 2010 NEWTEK"

You may permanently transfer all your rights under this License to another party by providing such party all copies of the Software licensed under this License together with a copy of this License and all written materials accompanying the Software, provided that the other party reads and agrees to accept the terms and conditions of this License.

### 2. Restrictions

The Software contains trade secrets in its human perceivable form and, to protect them, YOU MAY NOT REVERSE ENGINEER, DECOMPILATE, DISASSEMBLE, OTHERWISE REDUCE THE SOFTWARE TO ANY HUMAN PERCEIVABLE FORM. YOU MAY NOT MODIFY, ADAPT, TRANSLATE, RENT, LEASE, LOAN, RESELL FOR PROFIT, OR CREATE ANY MODIFICATIONS OR OTHER DERIVATIVE WORKS BASED UPON THE SOFTWARE OR ANY PART THEREOF.

### 3. Termination

This License is effective until terminated. This License will terminate immediately without notice from NEWTEK or judicial resolution if you fail to comply with any provision of this License. Upon such termination you must destroy the Software, all accompanying written materials and all copies thereof. You may also terminate this License at any time by destroying the Software, all accompanying written materials and all copies thereof.

### 4. Export Law Assurances

You agree that neither the Software nor any direct product thereof is being or will be shipped, transferred or re-exported, directly or indirectly, into any country prohibited by the United States Export Administration Act and the regulations thereunder or will be used for any purpose prohibited by the Act.

### 5. Limited Warranty and Disclaimer, Limitation of Remedies and Damages.

YOU ACKNOWLEDGE THAT THE SOFTWARE MAY NOT SATISFY ALL YOUR REQUIREMENTS OR BE FREE FROM DEFECTS. NEWTEK WARRANTS THE MEDIA ON WHICH THE SOFTWARE IS RECORDED TO BE FREE FROM DEFECTS IN MATERIALS AND WORKMANSHIP UNDER NORMAL USE FOR 90 DAYS FROM PURCHASE, BUT THE SOFTWARE AND ACCOMPANYING WRITTEN MATERIALS ARE LICENSED "AS IS." ALL IMPLIED WARRANTIES AND CONDITIONS (INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE) ARE DISCLAIMED. YOUR EXCLUSIVE REMEDY FOR BREACH OF WARRANTY WILL BE THE REPLACEMENT OF THE MEDIA OR REFUND OF THE PURCHASE PRICE. IN NO EVENT WILL NEWTEK OR ITS DEVELOPERS, DIRECTORS, OFFICERS, EMPLOYEES OR AFFILIATES BE LIABLE TO YOU FOR ANY CONSEQUENTIAL, INCIDENTAL OR INDIRECT DAMAGES (INCLUDING DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, AND THE LIKE), WHETHER FORESEEABLE OR UNFORESEEABLE, ARISING OUT OF THE USE OR INABILITY TO USE THE SOFTWARE OR ACCOMPANYING WRITTEN MATERIALS, REGARDLESS OF THE BASIS OF THE CLAIM AND EVEN IF NEWTEK OR AN AUTHORIZED NEWTEK REPRESENTATIVE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The above limitations will not apply in case of personal injury only where and to the extent that applicable law requires such liability. Because some jurisdictions do not allow the exclusion or limitation of implied warranties or liability for consequential or incidental damages, the above limitations may not apply to you.

### 6. General

This License will be construed under the laws of the State of Texas, except for that body of law dealing with conflicts of law. If any provision of this License shall be held by a court of competent jurisdiction to be contrary to law, that provision will be enforced to the maximum extent permissible and the remaining provisions of this License will remain in full force and effect.

### 7. Trademarks

NewTek, LightWave, LightWave 3D, LightWave CORE, LightWave 10, Viewport Preview Renderer, VPR, Virtual Studio Tools, TriCaster, TriCaster TCXD300, TriCaster TCXD850, 3PLAY, TriCaster PRO, TriCaster STUDIO, TriCaster BROADCAST, TriCaster DUO, SpeedEDIT, Virtual Set Editor, VSE, iVGA, LiveText, LiveSet, LiveMatte, TimeWarp, VT, [VTS], Video Toaster, Toaster, 3D Arsenal, Aura, are trademarks of NEWTEK. All other brand names, product names, or trademarks belong to their respective holders.

### 8. US Government Restricted Provision

If this Software was acquired by or on behalf of a unit or agency of the United States Government this provision applies. This Software:

- (a) Was developed at private expense, and no part of it was developed with government funds,
- (b) Is a trade secret of NEWTEK for all purposes of the Freedom of Information Act,
- (c) Is "commercial computer software" subject to limited utilization as provided in the contract between the vendor and the government entity, and
- (d) In all respects is proprietary data belonging solely to NEWTEK.

For units of the Department of Defense (DoD), this Software is sold only with "Restricted Rights" as that term is defined in the DoD Supplement to the Federal Acquisition Regulations, 52.227-7013 (c) (1) (ii).

Use, duplication or disclosure is subject to restrictions as set forth in subdivision (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at 52.227-7013. Manufacturer: NEWTEK, 5131 Beckwith Boulevard, San Antonio, TX 78249.

If this Software was acquired under a GSA Schedule, the US Government has agreed to refrain from changing or removing any insignia or lettering from the software or the accompanying written materials that are provided or from producing copies of manuals or disks (except one copy for backup purposes) and:

(e) Title to and ownership of this Software and documentation and any reproductions thereof shall remain with NEWTEK,

(f) Use of this Software and documentation shall be limited to the facility for which it is required, and,

(g) If use of the Software is discontinued to the installation specified in the purchase/delivery order and the US Government desires to use it at another location, it may do so by giving prior written notice to NEWTEK, specifying the type of computer and new location site. US Governmental personnel using this Software, other than under a DoD contract or GSA Schedule, are hereby on notice that use of this Software is subject to restrictions which are the same as or similar to those specified.

## Copyright and Trademarks

LightWave® 10

© Copyright 1990-2010 NewTek, Inc. All rights reserved.

Additional components Copyright: Steve Worley (1995-2010), Worley Labs (1995-2010), RainMaker (2003-2010) and Daisuke Ino (1995-2010)

FxMonkey™, HyperVoxels™, IntelligEntities™, Motion Mixer™, MultiMeshes™, Particle FX™, P.A.V.L.O.V.™, Skelegons™, SkyTracer 2™, Vmaps™ are trademarks of NewTek, Inc.

LightWave 3D is a registered trademark of NewTek, Inc. LightWave is a trademark of NewTek, Inc. All rights reserved. Windows and Windows Vista are registered trademarks of Microsoft in the United States and/or other countries. Intel and Intel Core are trademarks of Intel Corporation in the U.S. and/or other countries. AMD Athlon is a trademark of Advanced Micro Devices in the U.S. and/or other countries. Mac and Snow Leopard are registered trademarks of Apple Inc. NVIDIA and GeForce are trademarks of NVIDIA Corporation in the U.S. and other countries. All other trademarks are the property of their respective owners.



## Acknowledgements

---

### Engineering:

Chuck Baker, Deuce Bennett, Jamie Finch, David Forstenlechner, Matt Gorner, Mark Granger, Jonas Gustavsson, Bob Hood, David Ikeda, Antti Järvelä, Jarno van der Linden, Marvin Miller, Rob Powers, Prem Subrahmanyam, Jay Roth, Jon Tindall, David Vrba.

### Engineering Contributors:

Richard Brak, Andrew Cross, Josh Darling, Steve Forcucci, Art Howe, Peter Jespersen, Christopher Lutz, David Matthews, James Prentice, Mike Reed, Jarom Schow, William Vaughan, Scott Wheeler, James Willmott.

### Content Assembly:

Leigh Bamforth, Jarrod Davis, Lino Grandi, Luis Santos, Graham Toms, Gildardo Triana

### Installer:

Bob Hood, Dave Vrba

### Documentation:

Marvin Miller, Matt Gorner, Chuck Baker, William Vaughan

### Product Marketing:

Donetta Colboch, David Maldonado

### Special Thanks To:

Don Ballance, Steve Doubleday, Pat Grogan, Tim Jenison, Michael Kornet, Ralph Messana, Rex Olson, Jim Plant, Graham Toms, Andrew Yapple, Franck Lafage, Marc Gaillard, Hiroyoshi Abe, Ron Thornton, Jennifer Hachigian Jerrard, Dave Jerrard, Richard Morton, Reiner Shug, Lee Stringer and to all the Beta Team Members and HardCORE Participants

NewTek, Inc  
5131 Beckwith Blvd - San Antonio,  
TX 78249, USA  
1-800-862-7837 - +1-210-370-8000  
[www.newtek.com](http://www.newtek.com)

NewTek Europe  
Europarc - 17, Av. Léonard de Vinci  
33600 Pessac - FRANCE  
Tel.: +33 557 262 262 - Fax: +33 557 262 261  
[www.newtek-europe.com](http://www.newtek-europe.com)







# Table of Contents



## Contents

---

Software License and Limited Warranty .....	II
Copyright and Trademarks.....	III
Acknowledgements .....	III
.....	
Table of Contents.....	V



# Volume III . . . . . 1

## Surfacing & Rendering . . . . . 1

### Chapter 24: Surface Editor . . . . 3

Surface Editor . . . . .	4
Introduction . . . . .	4
Surface Editor Panel . . . . .	5
Texture Editor . . . . .	14
Layer Type: Image Mapping . . . . .	19
Layer Type: Procedural Texture . . . . .	24
Layer Type: Gradient . . . . .	34
Advanced Tab . . . . .	36
Environment Tab . . . . .	37
Shaders Tab . . . . .	38





# Chapter 25: Node Editor . . . . . 53

GUI Anatomy . . . . .	55
Menus and Hot Keys . . . . .	57
General Usage . . . . .	59
Connection Types . . . . .	62
About "Usage Examples" . . . . .	66
2D Textures . . . . .	67
Bricks2D . . . . .	68
CheckerBoard2D . . . . .	72
Grid2D . . . . .	74
Image . . . . .	77
NormalMap . . . . .	81
Parquet2D . . . . .	84
Planks2D . . . . .	87
Turbulence2D . . . . .	91
3D Textures . . . . .	95
Bricks . . . . .	95
Checkerboard . . . . .	99
Crackle . . . . .	101
Crumple . . . . .	104
Crust . . . . .	108
Dots . . . . .	111
FBM . . . . .	114
Grid . . . . .	117
HeteroTerrain . . . . .	120
Honeycomb . . . . .	123
Hybrid-MultiFractal . . . . .	127
Marble . . . . .	131
MultiFractal . . . . .	134
RidgedMultiFractal . . . . .	137
Ripples . . . . .	141
Turbulence . . . . .	143
Turbulent Noise . . . . .	147
Underwater . . . . .	150
Veins . . . . .	153
Wood . . . . .	156
Wood2 . . . . .	159
fBm Noise . . . . .	162
Constant . . . . .	165
Angle . . . . .	165
Color . . . . .	165
Direction . . . . .	165
Integer . . . . .	166



Pi.....	166
Purpose .....	166
Scalar.....	166
Vector .....	167
FUNCTIONS.....	167
Bias.....	167
BoxStep .....	168
Gain .....	169
Gamma.....	170
Modulate .....	171
Noise .....	172
Sine .....	173
SmoothStep .....	174
Wrap .....	175
Gradient.....	176
Tools - Incidence .....	176
Tools - Thickness .....	176
Gradient.....	177
ITEM INFO .....	180
Camera.....	180
Item Info.....	181
Light Info .....	182
Layers .....	183
Color Layer .....	184
Scalar Layer.....	185
Material Nodes.....	186
Conductor .....	186
Dielectric .....	188
Delta .....	190
Make Material .....	194
Math (Scalar).....	203
Abs.....	203
Add .....	203
BoxStep .....	204
Ceil.....	205
Clamp .....	206
Divide .....	208
Floor .....	208
Fresnel .....	209
Invert.....	209
Logic .....	210
Max .....	211
Min.....	211
Mod.....	212



Multiply .....	212
Pow .....	213
Sign .....	213
Smooth Step.....	214
Subtract .....	214
Trigonometry .....	215
ArcCos.....	215
ArcSin .....	215
ArcTan.....	216
Cos.....	216
Sin .....	217
Tan .....	217
Vector .....	218
Add4 .....	218
Add Scaled .....	219
Cross .....	220
Distance .....	220
Divide .....	221
Dot.....	221
Length .....	222
Multiply .....	222
Normalize .....	223
Scale .....	223
Subtract .....	224
Subtract Scaled .....	224
Transform.....	225
Transform2 .....	225
RayTrace .....	226
RayCast.....	226
RayTrace.....	226
Shaders (Diffuse) .....	227
Lambert .....	227
Minnaert .....	228
Occlusion.....	230
Occlusion II .....	231
OrenNayar.....	232
Theta.....	233
Translucency.....	234
Shaders (Reflection) .....	235
Ani-Reflections .....	236
Reflections.....	238
Shaders (Specular) .....	240
Anisotropic .....	240
Blinn .....	242





Cook Torrance .....	244
Phong .....	245
Shaders (Subsurface Scattering).....	246
Kappa .....	246
Kappa II .....	248
Omega .....	249
Shaders (Transparency).....	252
Ani-Refractions .....	252
Refractions .....	255
Spot.....	256
Spot Info .....	256
Tools .....	257
Color Scalar.....	257
ColorTool .....	258
Limiter.....	259
Make Color .....	260
Make Vector .....	260
Mixer.....	261
Vector Scalar.....	262
Vertex Map .....	262
Morph Map.....	262
Vertex Map .....	263
Weight Map .....	263
Nodal Glossary .....	264
Connection Table.....	266



<b>Chapter 26: Image Editor</b> . . . . .	<b>267</b>
Image Editor.....	268
Using the Image Editor.....	269
<b>Chapter 27: Rendering and Compositing</b> .....	<b>273</b>
Render Tab.....	274
Render Globals.....	274
Render Globals: Render Tab.....	278
Render Globals: Filtering Tab.....	281
Global Illumination Tab.....	282
Render Globals: Output Tab.....	288
Render Globals: Mask Tab.....	291
Rendering a Limited Region . . . . .	291
Render Frame . . . . .	292
Render Scene.....	292
Render Selected Object.....	292
Print Assistant.....	292
Network Render.....	293
Image Viewer.....	294
Compositing Options.....	295
Image Processing Options . . . . .	297
Image Processing: Pixel Filters.....	298
Image Processing: Image Filters.....	299
<b>Chapter 28: Distributed Rendering: Introduction</b> ....	<b>323</b>
Distributed Rendering: Introduction.....	324
Windows Setup.....	325
Mac OS X Setup.....	332
<b>Chapter 29: Preset Shelf</b> .....	<b>355</b>
Preset Shelf.....	356
<b>Chapter 30: Fiber FX</b> .....	<b>359</b>
What is FiberFX? . . . . .	360
How Fibers Are Created.....	360
FiberFX Pixel Filter Generated Fibers.....	361
Guide Geometry Based Fibers.....	361
3D Strand Modeler Geometry 'Fibers'.....	362
How Fibers Are Rendered.....	362









---

## Volume III

---

# Surfacing & Rendering

---





---

## Chapter 24: Surface Editor

---



## Surface Editor

### Introduction

With only a few exceptions, LightWave objects are composed of one or more polygons. What each polygon looks like is defined by its **surface attributes** or usually just **surface** for short. Groups of polygons can share the same surface and the polygons need not be contiguous. The polygons can even be in different objects.

### Surfacing Objects

Let's take a simple cube. It has six sides and, therefore, at least six polygons. Each side could have its own individual surface, all six sides could use the same surface, or any combination in between.



The polygons that make up a surface are given a name, usually in Modeler. The name can be changed, however, in the **Surface Editor**.

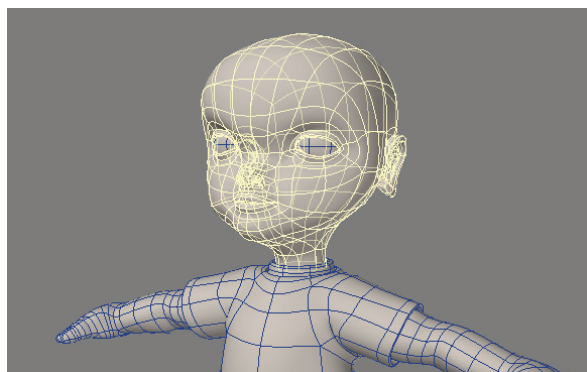
The surfacing information associated with the surface name is stored in the object file. When an object is later loaded, the surfaces used in it are also loaded, including any image files referenced in the surface attributes.



**WARNING:** **Surface** settings are saved as part of the associated object file(s), not in the Scene file. To save **Surface** settings, save the object.

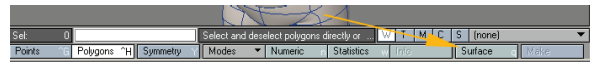
#### To assign surface names to polygons:

**Step 1:** In Modeler, select the polygons that you would like to surface.

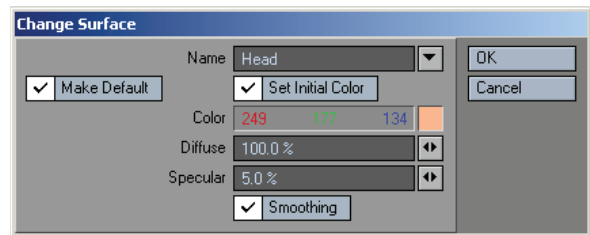


**HINT:** When you rename all of the polygons that already have a surface name, use the Statistics Panel (open/close with the Statistics button at the bottom of the interface - or press **w**) to select them. Also, if you are naming all polygons, you don't need to select any.

**Step 2:** Choose **Surface** (located at the bottom of the interface, or press **Q**) to open the **Change Surface** dialog.



Enter the surface name for the selected polygons and click **OK**. You can also select an existing surface name from the pop-up menu.

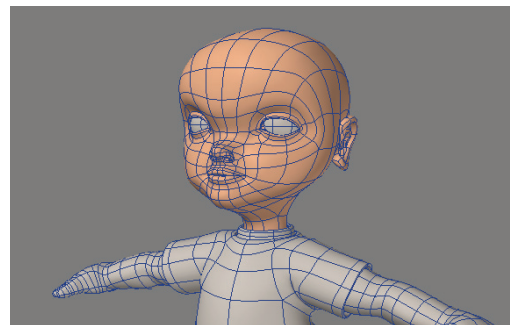


You should strive to give your surfaces meaningful names. For example, say you had a car object. One surface might be called "chrome," which you apply to the bumper, door handles, antenna, etc. "BlackRubber" might be another surface, which you apply to the tires, rubber gaskets, etc. You might use "RedMetal" for all of the exterior painted metal parts of the car.

To set some basic **Surface** settings, activate **Set Initial Color** and adjust the available settings as desired. You can perform more advanced surface editing using the **Surface Editor**.

The **Make Default** option automatically updates the **Surface** option on Modeler's **General Options Panel** (**Edit > General Options**). This controls the default surface name used when you create new geometry.

**Step 3:** Click **OK** to confirm.



### Surfacing Ideas

By no means should you restrict yourself to using, say, a procedural texture for what its name implies. The unlimited combination of settings, layers, and different surface attributes yields an infinite number of visual possibilities.

You are encouraged to load and render the various objects that come with LightWave. Study their surfacing techniques. Change them if you want. However, do not fall into the trap of thinking the surfaces are perfect. Many of them are far from it. Surfacing is an art form and 3D artists each have their own opinions on the best approaches. You need to develop your own.



## Surface Management

You can save surface attributes in a special surface file that you can re-use later. (Your **Surfaces** subdirectory should contain a multitude of pre-defined surface files.) Use the **Load** and **Save** buttons to do this. Use the **Rename** button to rename the surface.

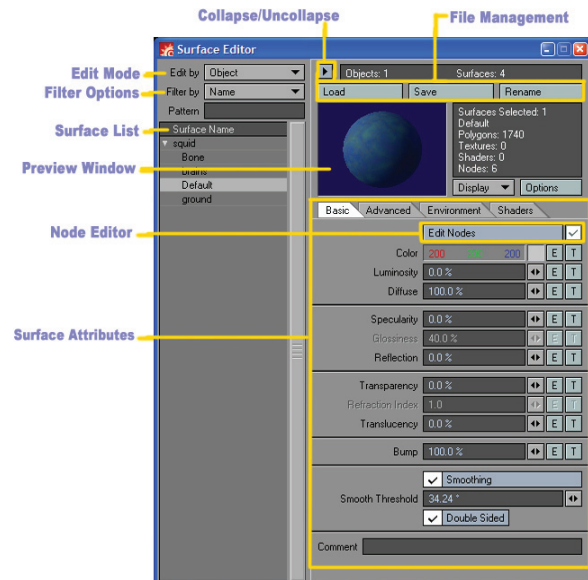


NOTE: When you load a surface file, remember that the surface texture **Scale**, **Position**, and so on are also loaded. Thus, if the object you apply the surface to is a different scale, you may need to tweak texture positioning data.

## Surface Editor Panel

(default keyboard shortcut **F5**)

The **Surface Editor Panel** will list the characteristics of your named surfaces; you can access this panel from both the Layout and Modeler modules.

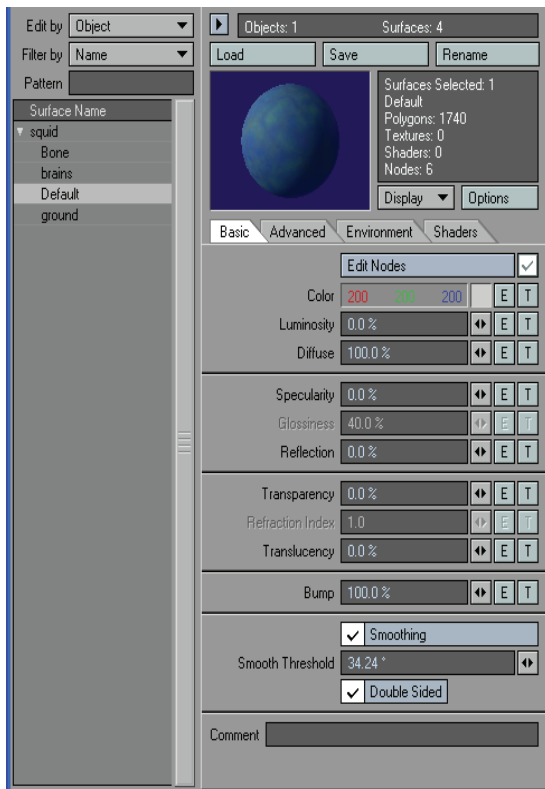


You can collapse/un-collapse the surface list by clicking the arrow button. When collapsed, a pop-up menu is added near the top to allow you to display and change the current surface.



## Compatibility Menu

Scenes created before LightWave v9.0 had Surface Editor features which did not work as intended. The features have been updated. The Compatibility menu allows you to use older scenes created before LightWave v9.2 and match those scenes rendered in older versions of LightWave. The Compatibility menu is found in the Advanced Tab of the Surface Editor.

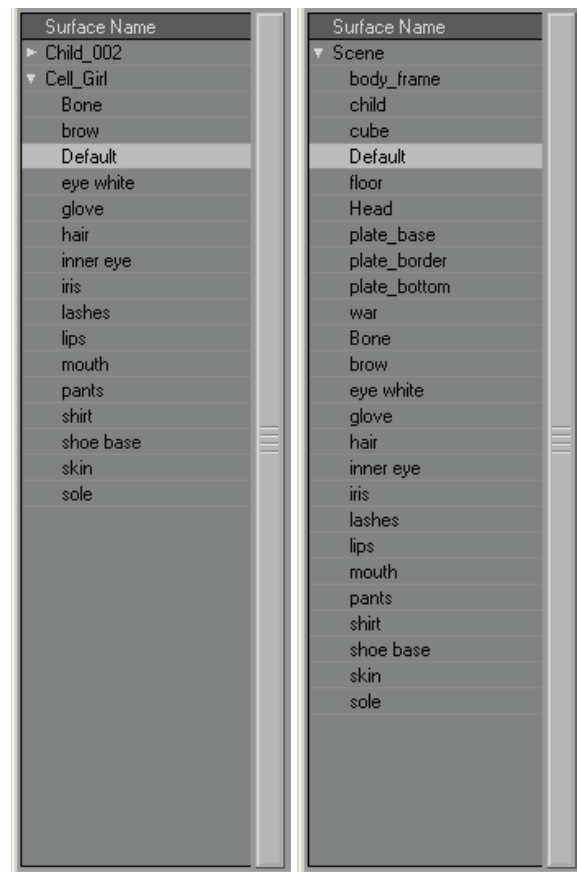


## Surface Edit Modes

LightWave offers two **Edit Modes** when you work in the **Surface Editor: Object** and **Scene**, which are **discrete** and **global Edit Modes**, respectively. These are selected using the **Edit by** pop-up menu. While surfaces are always saved within the object file itself, there are times when many objects share the same surface name. In **Object Mode**, each object's surfaces are protected from changes made on other objects with the same name. When in **Scene Mode**, all surfaces are **flattened**, so that any item in the scene that uses the current surface name receives the same changes to the surface attributes.

The **Edit Mode** is saved from session to session and affects how LightWave handles objects loaded with identical surface names. If the **Scene Edit Mode** is active, the last loaded object's surfaces will control.

When working in **Scene Mode**, you will notice that the **Surface** list shows only a **raw** list of surface names. While working in **Object Edit Mode**, you will see a complete list of the loaded objects with their surfaces listed beneath the object names.



Left: Object Mode, Right: Scene Mode

### Object Edit Mode

The default mode, **Object**, gives you many items in a scene with discrete **Surface** settings. For example, you may have two goblins, HOBGOBLIN and BOBGOBLIN loaded into a scene and each may have a surface called SKIN. In **Object Mode**, LightWave internally holds an object composite library for each item. So, while they both have a surface called SKIN, LightWave sees them as HOBGOBLIN/SKIN and BOBGOBLIN/SKIN so that you can make changes to either one without any risk of interfering with the other.

When you change from **Object** to **Scene Mode**, the last-loaded item determines the attributes for shared surface names.





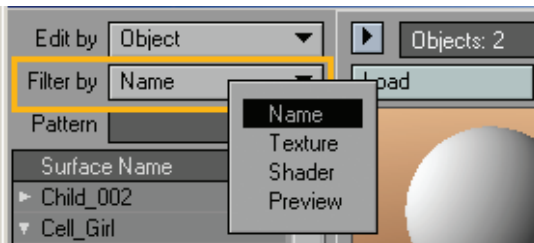
## Scene Edit Mode

The **Scene Mode** is very handy if you want many objects to share a surface. With this mode enabled, you can quickly make changes to a group of objects by changing their common surfaces. Internally, LightWave just drops the object composite library mentioned above, so that surfaces are referenced by their raw surface name, without object listing.

For example, you may have a troop of soldiers that all share a common uniform surface JUNGLE\_CAMO\_JACKET. Internally, LightWave simply manages the surface name JUNGLE\_CAMO\_JACKET rather than SOLDIER.01/JUNGLE\_CAMO\_JACKET, SOLDIER.02/JUNGLE\_CAMO\_JACKET, etc. If you want to change the base color of all soldier jackets, you can work in **Scene Mode** and make a single change that propagates throughout all items with a surface named JUNGLE\_CAMO\_JACKET.

## Surface List

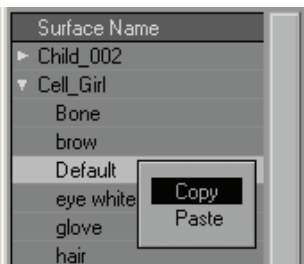
The large text window on the left lists the surfaces available for edit. The **Filter by** pop-up menu lets you filter elements shown in the surface list window. **Objects** shows all objects and their related surfaces. **Name** shows all surface names. **Texture** shows all surfaces that use procedural textures. **Shader** shows all surfaces that use surface shaders. **Preview** lists all surfaces that are visible in the image currently in the render buffer (you need to render a frame first). This is determined on a pixel-by-pixel basis. (Note that the **Preview** setting is only available if the **Surface Editor** is accessed from Layout.)



**Filter by** works in conjunction with the **Pattern** input field, which is a simple **include** filter. Any item that includes any of the text you enter will appear in the list window. You don't need to specify wildcards, just the text you want to include; when you leave the input field blank, you include all surfaces in the **Filter by** category. Once you have entered or cleared the text in the filter field, hit Tab to activate the filter.

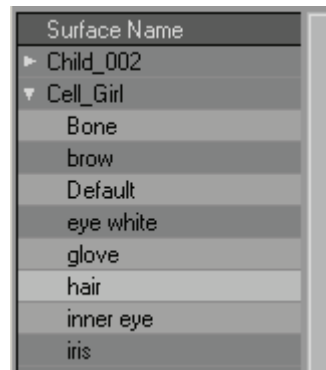
Select a surface to edit by clicking on its name in the list. If the object name is listed, you can expand or collapse the displayed surfaces by clicking on the arrow to the left of the object name.

When you right-click over the surface list window, a pop-up menu appears. You can **Copy** the selected surface to a memory buffer. Then, select a different surface and **Paste** the contents of the buffer over the settings of the selected surface.

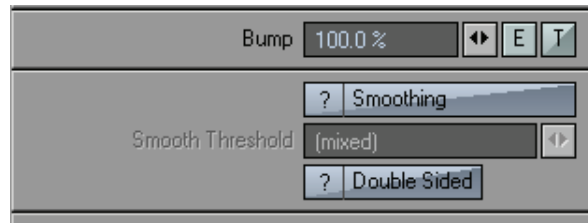


## Mass Surface Changes

You can select multiple surfaces in the **Surface Name** list and make mass surface changes. Hold the **Shift** key to select a range of surfaces or hold the **Ctrl** key to select/unselect surfaces independently.



Parameters with input fields that have different settings for the selected surfaces will show (mixed) in the field. If textures are different, the **T** button will appear in an intermediate state. Changing a surface attribute changes it for all selected surfaces, including **mixed** states. **Shift-clicking** on the **T** or **E** button removes the texture or envelope for all selected surfaces.



NOTE: Currently, you cannot make mass changes to surface shaders.

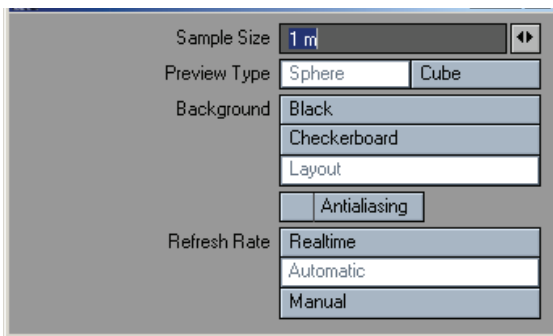
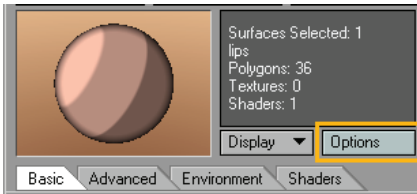


## Preset Shelf and VIPER

If you would like to use **VIPER** (Layout only) or the **Preset Shelf**, click their respective buttons on the interface.

## Preview Window

The preview window shows you surface samples for the active surface. You can access several **Preview** options by clicking the **Options** button.



**Sample Size** is the diameter of the sampled area on the surface. To obtain the most accurate preview, set this option to the approximate surface size to which you are applying the current attributes. **Preview Type** sets the shape of your sample surface.



You have several options for the preview **Background**.

**Checkerboard** is a good option for surfaces with some level of transparency. If you select **Layout**, it uses Layout's **Backdrop** settings. You can reduce **jaggies** in the preview by using the **Antialiasing** option.

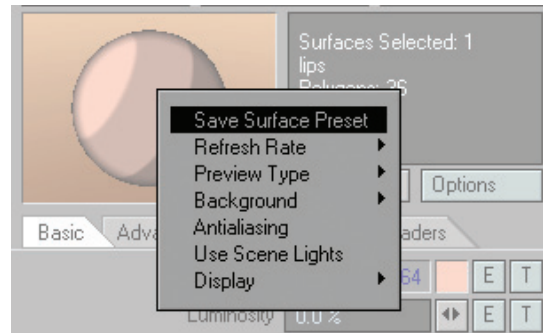
If **Use Scene Lights** is active, the lights from your scene will be used instead of the default preview light. This option is applied as if the preview object was at the Origin and also affects **VIPER**. Obviously, this is only available if the **Scene Editor** is open from Layout.

The **Refresh Rate** setting determines how the preview is refreshed when you make changes to **Surface** settings. If you set the rate to **Realtime**, the preview updates as you change interactive controls; for example, if you adjusted a mini-slider. When it is set to **Automatic**, the preview updates when you release the control. **Manual** requires you to click on the preview window to update.

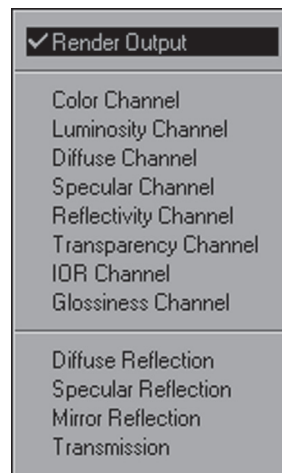
You can further tailor the preview to reflect only certain internal channels (buffers) instead of the normal **Rendered Output** with the **Display Mode** pop-up menu. This is very useful when you want to determine the effects on a specific channel. For example, if you apply a diffuse texture, you may want to see the effects of your settings on the actual diffuse channel.



NOTE: You can quickly set options by using the pop-up menu that appears when you hold the **RMB** over the preview window.



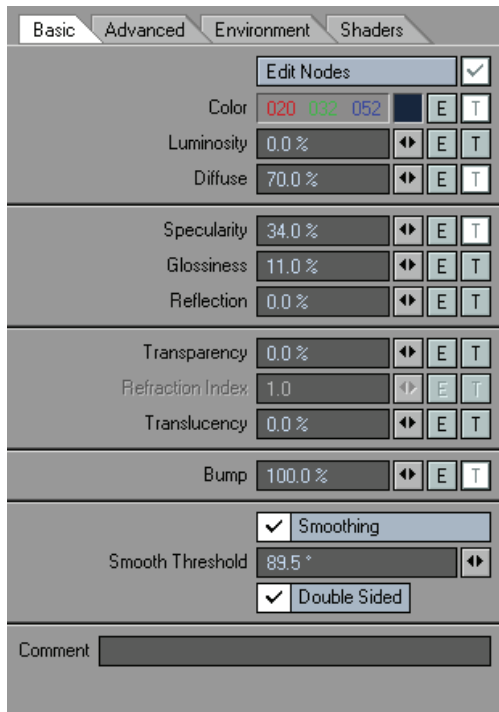
NOTE: Using the **Preview Options** pop-up menu gives you the added ability to view individual channels. Simply select **Display**, and the **Channel** you would like to preview.





## Basic Surface Parameters

The surface attributes on the **Basic Tab** are fundamental for virtually all objects. Each represents a certain characteristic of a surface's appearance. You may never have thought about it before, but if you look at the surface of objects around you, they differ by much more than their color. Are they shiny? Dull? Do they reflect other items in the room? Can you see through the material? These characteristics help you determine what materials objects are made of.



LightWave attempts to divide the different surface characteristics into controllable parameters.

### Edit Nodes

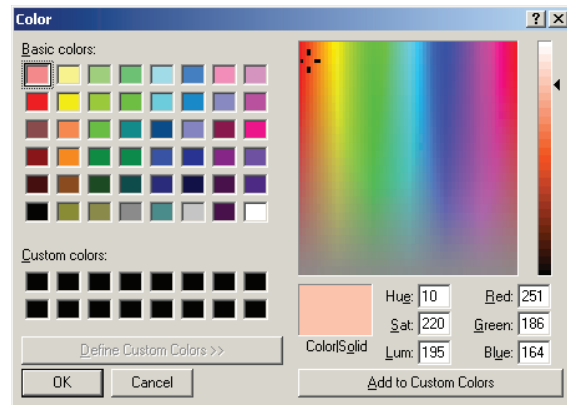
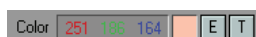
Activates the Node Editor. See the Node Editor documentation in Chapter 24 for more information.

### Numerical Settings

All of the **Numerical** settings have mini-sliders. You can enter a numerical value into the input field or use the slider. The range of the slider is based on realistic settings; however, the input field will accept values beyond the maximum and minimum possible with the slider, including positive and negative values. Keep in mind that, except for maybe **Luminosity**, values outside the normal range of 0 to 100 percent will be unrealistic.

### Color

**Color** is probably the most obvious surface parameter. It doesn't take much experience to know that if we want something to look like a banana, we need to make it yellow, right? However, since you are dealing with a 24-bit color palette and, thus, over 16 million colors, there are probably thousands of shades of yellow. Moreover, other settings, such as **Diffuse**, can have a dramatic effect on the final rendered color.



### Luminosity

**Luminosity** refers to how much a surface appears to glow of its own light. However, unless you use **Radiosity**, **Luminosity** does not have any actual light emitting properties — you need to add an actual light for that. A value of 0% is most common for this setting, unless a special need arises such as the surface of a modelled light bulb. This surface property is not the same as **Glow** on the **Advanced Tab**.



Left: Luminosity 0%, Right: Luminosity 100%

### Diffuse

**Diffuse** (sometimes called **diffusion**) is the amount of light scattered by a surface. A high level scatters a lot of light, and therefore, the surface appears bright. A low level absorbs most of the light, and therefore, the surface appears dark and dull. Metal and dirt surfaces are good candidates for a low **Diffuse** level. Common values are 40% to 80%. Surfaces must have some diffusion for shadows cast on them to be visible.

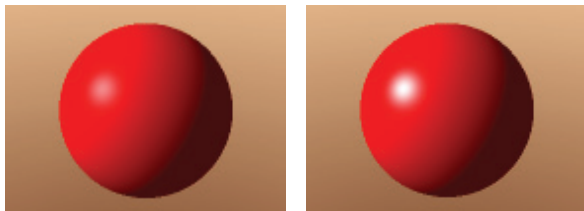


Left: Diffuse 100%, Right: Diffuse 50%



## Specularity

**Specularity** is a kind of reflection or highlight that occurs on the surface of smooth or shiny objects. This highlight is really the reflection of the light source. **High Specular** levels are commonly used on glass spheres, chrome bumpers, and so on. How the surface reflects this highlight tells the observer if the surface is dull, smooth, shiny, hard, or even metallic. Generally, the highlight assumes the color of the light source that causes it, but you may change this with the **Color Highlights** settings in the **Advanced Tab**.



Left: Specularity 50%, Right: Specularity 100%

## Glossiness

When some level of **Specularity** exists, **Glossiness** determines how a highlight spreads out. A low setting creates a large highlight, whereas a higher setting creates a smaller highlight.



Left: Glossiness 5%, Right: Glossiness 50%

## Reflection

**Reflection** determines how much a surface shows a reflection of its surroundings. A mirror is an example of a highly reflective surface. LightWave will ray trace reflections or you may use options that fake them, which can substantially reduce rendering time.



Reflection 25%



Reflection 100%



## Transparency

**Transparency** lets you make a surface **see-through**; it is the opposite of **Opacity**. Whenever light passes through a transparent object, the light will bend. An example of this can be seen when you look through a glass of water or down through a clear pool of water. The amount of this light bending is controlled by the **Refraction Index**.



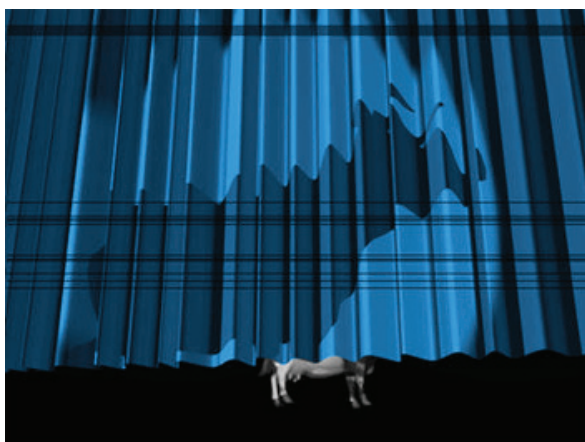
Image by Christophe Desse



NOTE: To control the amount of **Edge Transparency**, you must add the shader of the same name.

## Translucency

**Translucency** allows back lighting. The effect can be seen when someone stands behind a curtain and you see their silhouette through the material. Other examples include seeing bones in your hand with a flashlight in your palm, or bugs crawling on the underside of leaves.



**Translucency** is similar to **Transparency** in that all lighting properties, like color and luminosity, will show through. The obvious difference is that translucency doesn't add a **see-through** effect.

If you want to make a silhouette visible, something must cast a shadow on the back side. Note that you do not need rear-facing polygons for the back side, nor must you use a double-sided surface to **catch** the shadow.

## Bump Map

Nearly all real world surfaces have some amount of texture or bumpiness. Such bumps are apparent due to the way light falls across the surface. A **Bump Map** is applied to a surface to give the appearance of a texture. However, no real geometry is added and if you looked across the surface from the side, the illusion could be ruined. Shadows and edges can also reveal this lack of geometry in certain circumstances. The default setting is 100%.



No Bump



Bump on Skin

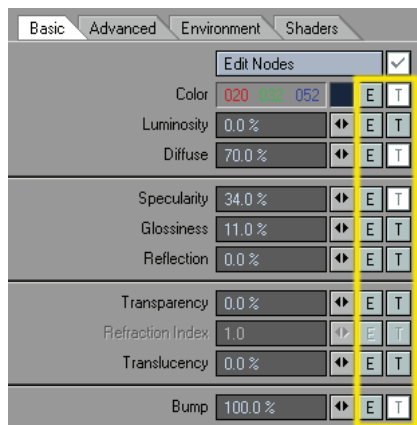
If you want your bumps to actually displace geometry, try the **Bump Displacement** option (**Object Properties**).





## Envelopes and Textures

Very few things in the real world have constant surface properties. Color patterns, discoloration, nicks, scratches can all subtly contribute to an object's appearance. The real world is anything but consistent, but that is what makes things look **real**. The **E** and **T** buttons let you use envelopes and textures, respectively, instead of a static numerical value. The proper use of these features often results in a much more realistic surface.



Envelopes let you vary a value over time. For example, instead of Luminosity having the same value throughout your animation, it can differ on each frame. However, in any particular frame, it will have that value over the entire surface. To vary the value over the surface area, you must use a Texture.

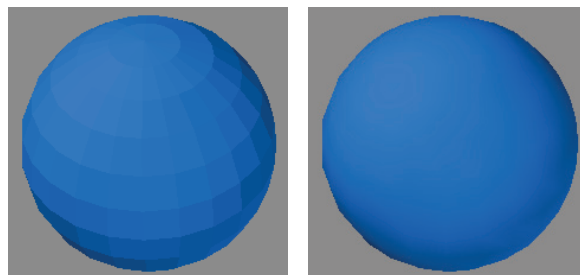
Envelopes use the **Graph Editor**, which is discussed in depth in Chapter 6 -**Graph Editor**, starting on page 179.



NOTE: Envelopes are not limited to surface values, so you will find **Envelope E** buttons throughout LightWave.

## Surface Smoothing

**Smoothing** causes objects to appear to have smoothly rounded surfaces even though the object is composed of flat-faced polygons. To do this, LightWave uses a technique known as **phong shading**. If the edges of two smooth-shaded polygons share vertices (points), they appear as one continuous, curved surface. The shared edge between them is no longer visible.



Left: No Smoothing, Right: Smoothing

### Smooth Threshold

By default, LightWave will not smooth across two polygons if the angle between them is 90 degrees or sharper, unless you adjust the **Smooth Threshold**. This value adjusts the range of the smoothing function; it defines the angle beyond which LightWave will not smooth over the seam between adjoining polygons. The angle is measured using the surface normals of the adjoining polygons. If this angle is less than the **Smooth Threshold**, the surfaces are rendered smoothly. The default of 89.5° assures that any surfaces at right angles (90°) or greater to each other are not smoothed.

Sometimes, due to the way an object is modelled, you may get unexplained rendering errors on smoothed surfaces. Try increasing the **Smoothing Threshold** to correct such errors. We recommend using smaller increases first, although for extreme displacement-mapped objects, you may try fairly high values if rendering produces a mixture of jagged and smoothed results.



NOTE: As with **Bump Mapping**, **Smoothing** does not actually change the surface's geometry. As such, the edges of a ball can still expose the straight polygon nature of the object. If this becomes a problem, make the actual geometry smoother. You can use Modeling tools like **Metaform**, for example.

### Sharing Points

The concept of sharing points is important in surface smoothing. Unlike the real world, objects, and their points, can occupy the exact same space. A simple cube has eight points and six sides. Any one point is shared by three sides. However, you can create a seemingly identical cube that doesn't share any points or sides. Each side would be a totally independent polygon, thus there would be 24 points (six sides times four points).

The obvious reason for using shared points and sides is that it creates a more compact object. Remember, the simpler the object, the faster it renders and the lower the amount of resources used (i.e., RAM). However, sometimes you will find that the polygons need to be separated.



## Using Separation

**Surface Smoothing** can occur only where two polygons share **all** of the points along an edge. However, sometimes this smoothing is not what you want. For example, if you were Modeling something where you wanted a visible physical break, like a seam, you'd want to cut the polygons away and then paste them right back in. You may also find that when you have a flat surface that rounds over a corner, separating the flat polygons from the initial rounded polygons gives a more realistic look.



Left: Surface Smoothing, Right: Surface Smoothing with Separated Polygons

If you examine objects around you, you should notice that they all have seams. Very few objects in the real world are perfectly smooth with edges that align. Separating polygons can add subtle detail to objects and give them a real world look.



**NOTE:** Separating polygons will double the points along the previously shared edge since each polygon now has its own set of those points.

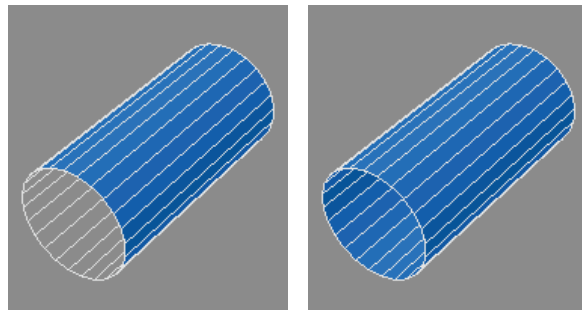


**HINT:** Using the shortcut keys, **Ctrl X** and **Ctrl V**, to cut and paste, is a quick way to separate polygons.

This technique lets you control smoothing by using a single surface name. Although you could create independent surfaces with identical settings — except one has smoothing on and the other does not — you may prefer to have one with smoothing on and use this separation technique to create the seam.

## Double Sided

Sometimes it's necessary to see both sides of a polygon, such as when you want to go inside a model (and wall thickness is not a concern). Also, if you import objects from other 3D modeling programs, either directly or through a conversion utility, some polygons may end up facing the wrong direction, which causes the object to render improperly. Click **Double Sided** to treat all polygons on the selected surface as two-sided.



Left: Single-Sided Polygons, Right: Double-Sided Polygons

As a consequence of using **Double Sided**, rendering time will increase because LightWave must calculate more polygons. Note also that **Double Sided** is only a surface attribute. The object, when loaded into Modeler, will show the true single-sided polygons.

## Comment

You can add a comment to the surface in the **Comment** field.

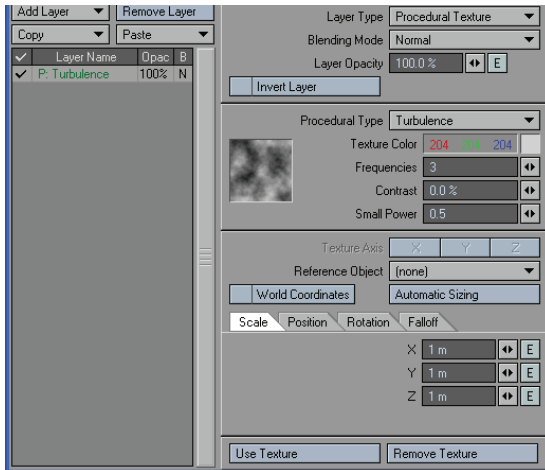
Comment





## Texture Editor

While you can vary a parameter over time with an envelope, the parameter is not dynamic over the surface: the value for a particular frame is the value for the entire surface. Textures, on the other hand, essentially let you vary the value dynamically over the surface (as well as in time). A **Color** texture is probably the easiest illustration. Instead of using the same color on a surface, you can map a picture (a color texture **image map**) onto the surface that lets the color differ over the surface area.

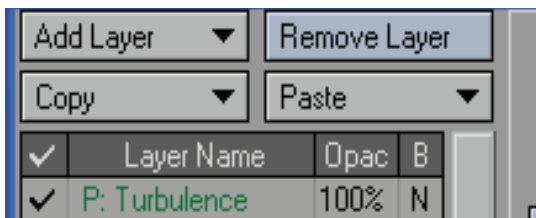


You can use **Image Maps** for more than just color. On other surface parameters, the color element is ignored and only the brightness of each pixel varies the value over the surface. For example, you can use an **Image Map** with **Specularity** to suggest scratches on a surface. Textures let you divide a surface into thousands of smaller areas and apply a different level of the surface attribute to each individual area.

In addition to **Image Map** textures, you can also use **Procedural Textures** and **Gradients**. **Procedurals** are more or less mathematically computed images. **Gradients** act as a type of envelope, except that the value can change over a condition other than time. Select these options by using the **Layer Type** pop-up menu.

### Texture Layers

LightWave can layer surfaces infinitely; once you set the original texture, you can add another layer by using the **Add Layer** pop-up menu. You can choose to add any of the layer types — the type can be changed later if necessary.



The layer list window shows you all of the layers. The first column indicates whether the layer is active (checked) or not (blank). You can toggle the status by clicking in the column. The next column shows the layer name, which indicates the type of layer. Next is the opacity percentage, followed by the **Blending Mode** (e.g., + for Additive).

You can choose which layer to work on by simply clicking on it in the window. To remove a layer, select it and then click **Remove Layer**.

### Copy and Paste

You can copy the currently selected layer or all of the layers in the Surface list to a memory buffer by using the **Copy** pop-up menu. For the **Paste** operations, **Replace Current Layer** will replace the selected layer with the buffer contents, even if the buffer contains more than one layer. **Replace all Layers** will clear all existing layers and then add the buffer contents. **Add to Layers** simply appends the buffer contents to the end of the list.



NOTE: The buffer contents will remain until you quit LightWave.

### Layer Order

Texture layers are always listed in order from top to bottom. New layers are added on top of existing layers. Keep this in mind when you set up multiple layers. For example, if you want an **Image Map** with fractal noise across the entire surface including the image, you must map the image first and add the fractal noise afterwards. Of course, you can reorder layers by dragging them with your mouse.



NOTE: The base **Surface** settings — those set on the **Surface Editor's Basic Tab** — always sit beneath all layers.

### Blending Layers

To set the opacity of a texture, use the **Layer Opacity** field. Reducing the value from 100% makes the overall texture more and more transparent. Setting it above 100% can force a texture layer to unnatural values.

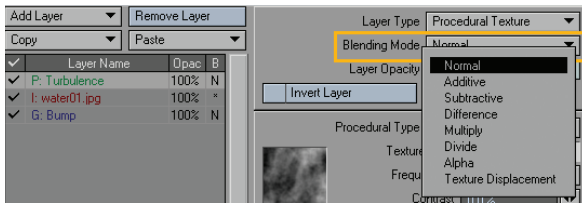
The **Blending Mode** determines how the layer is blended with other layers. With **Normal**, underlying layers will be totally covered (i.e., **replaced**) by the texture, assuming **Layer Opacity** is 100%. The texture acts like an alpha matte; thinner areas of the texture allow the underlying layers to show through. If **Layer Opacity** is 50%, you get 50% of the layer and 50% of the underlying layers. **Additive**, adds the texture (times the **Layer Opacity**) to underlying layers.



NOTE: In versions prior to 7, "Normal" was called "Additive." The current **Additive Mode** was new for version 7.

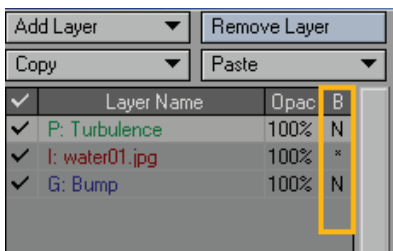
To achieve an even blend between multiple texture layers, use **Normal**. Then divide the number of the layers into 100% and use the resulting value as your **Texture Opacity** value. For example, the first (bottom-most) layer would be set to 100% (100/1), the second layer will be set to 50% (100/2), the third layer will be set to 33% (100/3) and the fourth layer will be set to 25% (100/4).

**Subtractive** subtracts the layer from the underlying layers. **Difference** is similar to **Subtractive** but takes the absolute value of the difference. **Multiply** factors the layer by the underlying layers. Multiplying by darker colors will darken the image, while brighter colors will brighten. **Divide** multiplies the underlying layers by inverse of the layer. This generally has the opposite effect of **Multiply**.



The **Alpha Blending Mode** makes the layer an alpha channel on the preceding layer. In other words, it **cuts out** parts of the preceding layer and makes those areas transparent. White in the alpha image creates opaque areas and black creates transparent areas. Shades in between will do a little of both. If the image, procedural, or gradient has color, the alpha image is based on the brightness of the areas.

Each layer's **Blending Mode** is indicated in the right-most column of the layer list.



**Texture Displacement** displaces (distorts) layers above it, similar in effect to a **Bump Map**.

### PShop Filters

Photoshop-style filters have been added to the Blending Mode menu in the Texture Editor. They are designed to work like the blending modes found in Photoshop. While the names may be similar in some instances, such as LightWave's native Multiply and the Pshop Multiply mode, the Photoshop modes are mathematically different from the native LightWave blending options, and therefore produce different results.

**PShop Multiply**—Blends the base color with the blend color, making the image darker by multiplying the color values together. A black color will make the final blend black. A pure white color will change nothing. Working with colors other than black or white will produce a darker color.

**PShop Screen**—The opposite of PShop Multiply, the color values are inverted, multiplied together, then inverted again. A black value will change nothing, while a white color will produce a white result.

**PShop Overlay**—A combination of PShop Multiply and PShop Screen, the formula is dependent on the base color. A lighter base color will produce a lighter result, while a darker base will result in a darker color.

**PShop Softlight**—Similar to PShop Overlay, except the blend color is the determining factor. A lighter blend color will produce a lighter result and a darker blend color will produce a darker color. Colors are mixed, so a pure white or pure black blend color does not produce a result of pure white or pure black.

**PShop Hardlight**—The opposite of PShop Overlay (not PShop Softlight!) and is dependent on the blend color. Multiplies the blend and base colors, with a lighter blend color resulting in a lighter result and dark blends producing darker results.

**PShop Colordodge**—The base color is brightened to reflect the blend color by reducing the contrast. A blend color of black changes nothing.

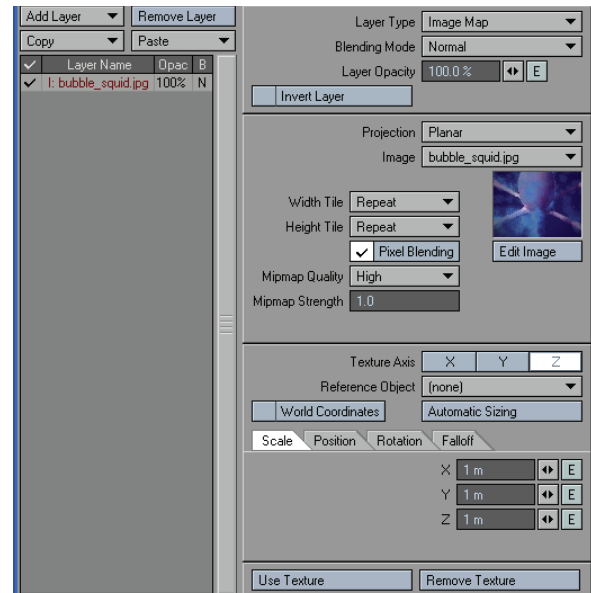
**PShop Darken**—The base and blend colors are compared and the darker color is the one used.

**PShop Lighten**—The base and blend colors are compared and the lighter color is selected.

**PShop Exclusion**—Subtracts the blend color from the base color, or base from blend, whichever has the brighter color value. A pure white blend value will invert the base color value.

### Image Properties

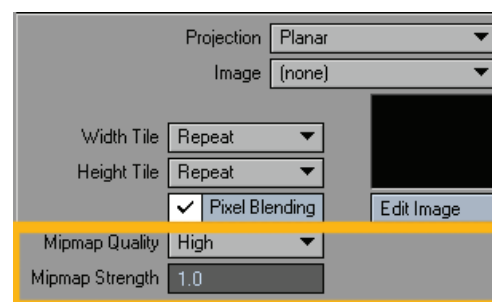
After you select one of the **Projection** settings, use the **Image** pop-up menu to select or load an image/sequence. Clicking the **Edit Image** button launches the **Image Editor Panel**. Here you can adjust certain aspects of the image(s).



The available **Image** settings on the **Texture Editor Panel** vary depending on the **Projection** setting. Two settings always appear, however. **Pixel Blending** smoothes out the pixelisation that can occur when a camera gets close to the mapped surface.

### OpenGL MipMap

Mip-mapping is used in LightWave and today's games to avoid texture graininess at a distance or at a low angle. Basically, lower-res versions of the texture are generated in real time to save time processing the original image to map onto a polygon. Please note that due to the nature of this filtering method, low-res textures may appear somewhat blurry. Mip-mapping is very useful in animation projects because it ensures that the same image is used no matter where the image-bearing object is in the scene, however, the total memory cost for mip-mapping tends to average out at 1.5 times the image size in memory. On still renders, you can turn off mip-mapping to save memory.





The antialiasing **Strength** value determines the amount of antialiasing. The default setting of 1 should be used in most cases; however, the level may need to be raised or lowered slightly to better match the particular image. This value can be set to higher levels to add blurring effects to the image.

**Planar, Cubic, and Cylindrical Image Map** projection have **Tiling** (i.e., repeating) options, which you can set independently for the horizontal and vertical repeating directions.

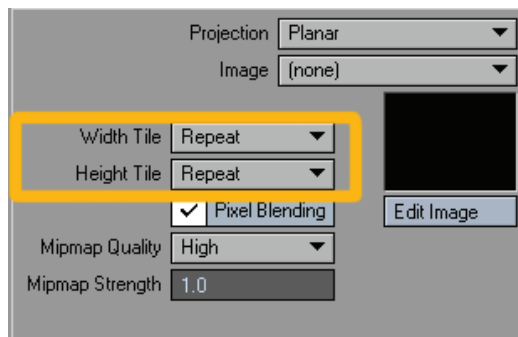
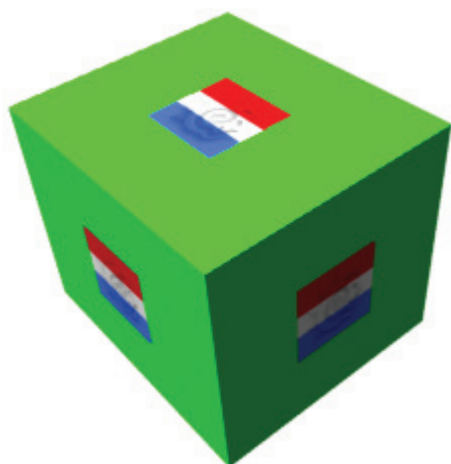


Image used for the following examples

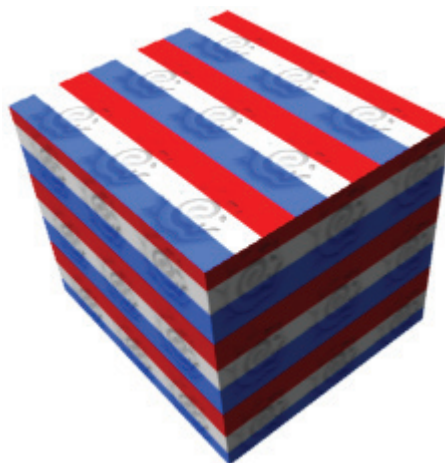
### Reset

**Reset** is the **no-repeat** mode. The underlying surface will be visible if the image is smaller than the surface area.



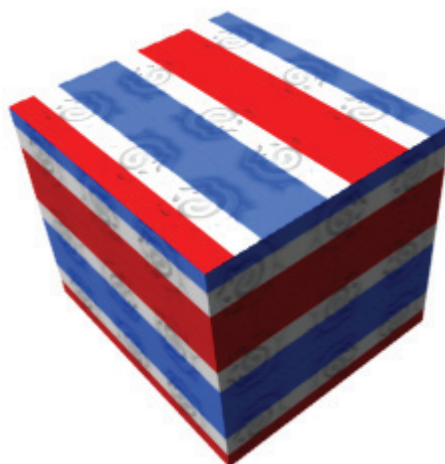
### Repeat

**Repeat** tiles the image until it fills the surface.



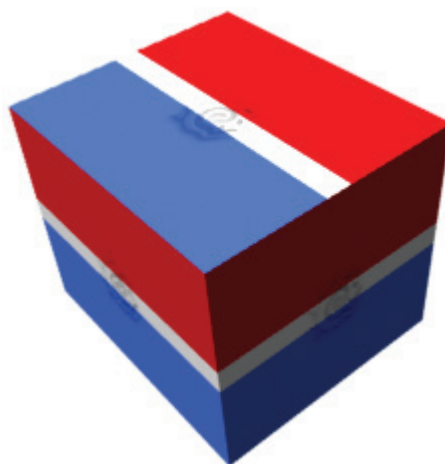
### Mirror

**Mirror** tiles the image, but flips the image horizontally or vertically.



### Edge

**Edge** extends the edges of the image to cover the surface. This setting works best when the outer edges are a solid color.



NOTE: The repeating options are relevant only if the **Scale**, **Position**, and **Rotation** settings (at the bottom) are set in such a way that the image will not fill the surface area.



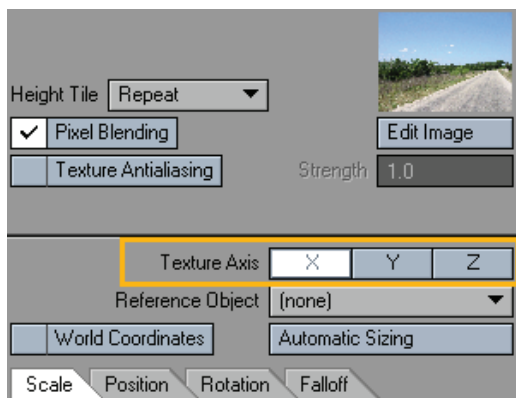
**Cylindrical** and **Spherical** have **Wrapping** options that set how many times the image appears horizontally, **Width Wrap Amount**, and vertically, **Height Wrap Amount** (not applicable to **Cylindrical**). This sets how many times you want the image wrapped for the given **Scale** values. Generally, you'll want this set at 1.



NOTE: Wrap amounts can be negative, which will reverse the image as it is wrapped around in the opposite direction.

## Texture Placement

Once you set the image properties, you must now scale and position the image. Textures are initially positioned referenced to the **X**, **Y**, or **Z Texture Axis**. For **Planar** projection, think of the axis as a nail used to hold a photograph to a wall. For **Cylindrical**, think of the axis as the cardboard center of a roll of paper towels. **Spherical** is very similar. However, you cannot select an axis for a **Cubic** map.



For example, a soda can would use a **Y Texture Axis** because it sits vertically. The fuselage of an airplane, on the other hand, would probably use the **Z Texture Axis**.

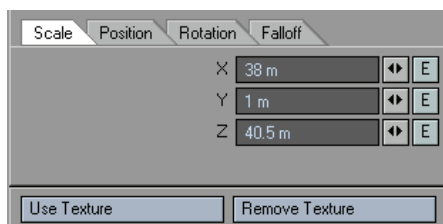


NOTE: You should generally model objects so that they face the positive Z axis.

If the **Texture Axis** is perpendicular to a surface (such as the sides of a box using the same surface name as the **projected** side, the image will run **through** the surface.

## Surface Size, Position, and Rotation

**Scale** defines the size of the surface to which the texture is applied. **Position** defines the coordinates of the center of the texture on the surface. **Rotation** defines how much the texture rotates around the center **Position**.



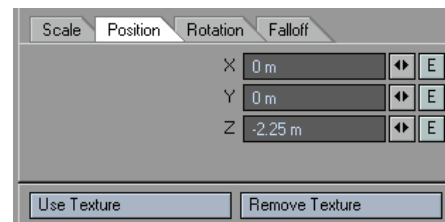
NOTE: Different combinations of **Texture Axis** and **Rotation** can achieve the same results.

Generally, you want to match these settings with the actual surface. **Automatic Sizing** causes LightWave to calculate the **Scale** and **Position** of a selected surface and input those values. It computes an imaginary bounding box around all areas using the current surface. This will fit an **Image Map perfectly** onto the surface.



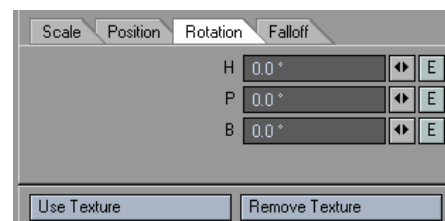
HINT: Use Automatic Sizing for starting values when using procedural textures.

If the **Scale** is smaller than the surface size, an **Image Map** will tile across the surface (assuming that one of the **Repeat** options is active). You might also see the tiling effect if the **Position** is not set to the actual surface center.



Because LightWave's textures are three-dimensional, there is no rule stating that the center of a texture must be located somewhere within the bounds of the surface. You can place the texture center outside of the surface and still get a texture to appear across the surface. Textures **extend** in all directions throughout the virtual space in Layout, but appear only on the surfaces that are set to display them.

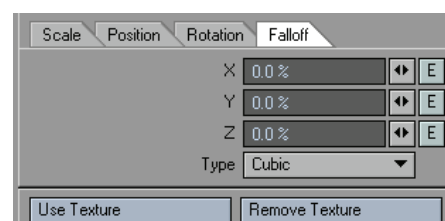
There are definitely times where you don't want the texture sized exactly to match the surface. Such is the case with **Procedural Textures**, which usually look better when the **Scale** is smaller than the **Surface Size** because it adds more detail. Another example is when you use only part of an image that is texture mapped.



## Falloff

The **Falloff** values specify the apparent falloff for every **unit of distance** moving away from the texture center, defined by the **Position** settings. (The unit of distance is defined by the **Default Unit** setting on the **General Options Tab** of the **Preferences Panel**.) When you animate the **Position** with an envelope, the position at frame 0 is used as the center for falloff purposes.

The **Type** pop-up menu determines the shape of the **Falloff**. (In previous versions, the **Falloff** was always cubic.) **Linear Falloff** types (i.e., **LinearX**, **LinearY**, and **LinearZ**) operate only in one direction. If you use 100% **LinearX**, for example, the texture will fall off only in the positive X direction from the texture center. To fall off towards negative X, use -100%.







With **Cubic**, the **Falloff** occurs on both the positive and negative sides. You can achieve a linear-type falloff in both directions by using **Cubic** and setting the two axes you do not wish to fall off to 0%.

## Using the Reference Object

Although you can explicitly set the **Position**, **Size**, and **Rotation** of a texture, you can also assign a **Reference Object** — normally a **Null** object — and get similar results. The **Reference Object** makes all of these settings operate relative to itself. You can better control the texture animation by animating the **Reference Object**. Moreover, you can use different **Reference Objects** for different **Surface Textures** (e.g., surface color, diffusion, specular, and so on).



NOTE: An object's pivot point does not affect surfacing.

## Freezing Reference Object/Camera

If you set a **Reference Object** (or **Reference Camera** used with **Front Projection Mapping**) and then select **(none)** from the related pop-up menu, a dialog appears that asks, "Do you want to keep item's parameters?"

If you click **Yes**, you will store the reference item's state at the current frame in the object. (Don't forget to save it.)

## World Coordinates

Normally, textures are **locked** to a surface and travel with it, as the object is moved, stretched, rotated, or deformed. Selecting **World Coordinates** will lock the texture to LightWave's Origin instead of those of the surface. Moving the object now will result in the object traveling **through** the texture. All **Texture** settings continue to be valid, but now are relevant to the **World Coordinates**.

**World Coordinates** can create the look of light beams moving through a fog. You create this effect by using transparent light beams with a fractal noise texture for **Color** and/or **Luminosity** activated. As the light beam is moved about, it will move through the fog created by the fractal noise. Using **World Coordinates** on a **Transparency Map** with **Falloff** can make a spaceship **cloak** (turn invisible).



## Layer Type: Image Mapping

Except for the **Color** surface attribute, when you use image maps the brighter the data, the higher the setting, and the darker the data, the lower the setting. A totally white image is the same as a 100 attribute value and a totally black picture is the same as a 0 attribute value. As such, if you use a standard RGB picture, you cannot get values higher than 100 or less than 0, but it will usually contain a wide array of brightness values. However, if you use a high dynamic range image, you can exceed this limit.



NOTE: To see image maps in the Layout view, make sure the viewport is set to **Textured Shaded Solid**. You can change this on a viewport's titlebar.

### Image Map Projection

Since images are usually rectangular and surfaces may or may not be, you must tell LightWave how you want the image map **projected** onto the surface. The common projection types settings are **Planar**, **Cylindrical**, **Spherical**, **Cubic**, **Front** and **UV**.

Generally, you should pick the shape that best describes the surface shape. (Note that this is not necessarily the object's overall shape since that may be made up of many surfaces.) For example, if you were mapping a label image on the sides of a soda can, you'd use **Cylindrical**. For a planet, you'd use **Spherical**. For a wall, **Planar** would do the trick. For a brick, **Cubic** might be best.

What about a die? **Cubic**? This may be a trick question. Since a die has a different number of dots on each side, you'd use a different **Planar** map on each one.

Standard **Image Mapping** tools (i.e., **planar**, **cylindrical**, and **spherical mapping**) may be somewhat limiting when the surface is irregular in shape. These techniques usually work well only in cases where the entire texture image can be globally mapped using a linear interpolation along two axes. The object geometry essentially has no influence on how the texture is applied.

However, what if you could assign areas of a texture image to points on the surface, essentially **tacking** it down at key points? This is what **UV Mapping** in Modeler allows you to do. Between the tacks, the image is stretched smoothly.

## Planar Projection

**Planar** projection will project an image onto a surface as if you were projecting the image through a slide projector onto a wall. **Planar Image Maps** are best used on flat, or nearly flat surfaces like the sides of buildings and the screens of video monitors.

Planar Image Mapping	Axis / Examples	Diagrams
	X  Mapping insignia onto the tailfin of an aircraft.	
	Y  Mapping aerial photo onto terrain.	
	Z  Mapping a picture onto a TV Screen.	

For the X and Y axes, **Planar** images are **projected** from the positive axis side of a surface towards the negative axis side. This means that the image appears correct when viewed from the positive side and it appears reversed if you view it from the negative side. For the Z axis, **Planar** images are projected from the negative side.


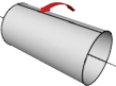

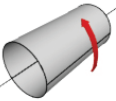


HINT: If you encounter this reversing effect and it isn't what you want, you can reverse an image back by making the **Scale** value negative for the axis that needs to be reversed.



Cylindrical Projection


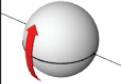

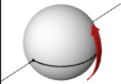
**Cylindrical** projection wraps an image around the selected axis like a paper towel wrapped about a cardboard tube. By default, an image is wrapped so it covers the surface once, allowing the side edges of the image to meet on the **back** of the surface. A soda can and a tree trunk are both good examples of surfaces that would use **Cylindrical** projection.

Cylindrical Image Mapping	Axis / Examples	Diagrams
	<b>X</b> Mapping detail to a pipe segment.	
	<b>Y</b> Mapping a label onto a soda can or a wine bottle.	
	<b>Z</b> Mapping detail onto an engine exhaust.	

**Cylindrical** projection is always wrapped around a surface so that the top of the image appears towards the positive axis side of the **Texture Axis**.

Spherical Projection

**Spherical** projection wraps an image around a surface as if you were stretching a flat piece of rubber around a ball, but without having to worry about the edges all meeting. Planets, basketballs, and marbles could all use **Spherical** projection.

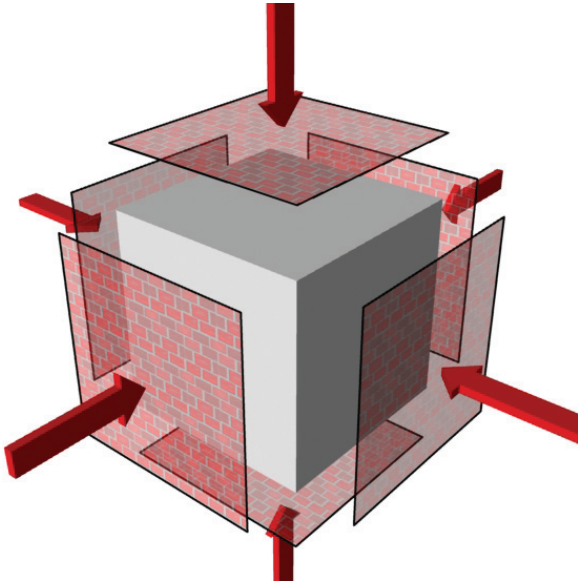
Spherical Image Mapping	Axis / Examples	Diagrams
	<b>X</b> Rarely used.	
	<b>Y</b> Wrapping a map to create a planet, or a basketball.	
	<b>Z</b> Rarely used.	

**Spherical** projection does not use **Scale** parameters. Images are wrapped completely around the surface (using the **Wrap** values, discussed later). **Spherical** projection is always oriented so that the top of the image appears toward the positive side of the **Texture Axis**.



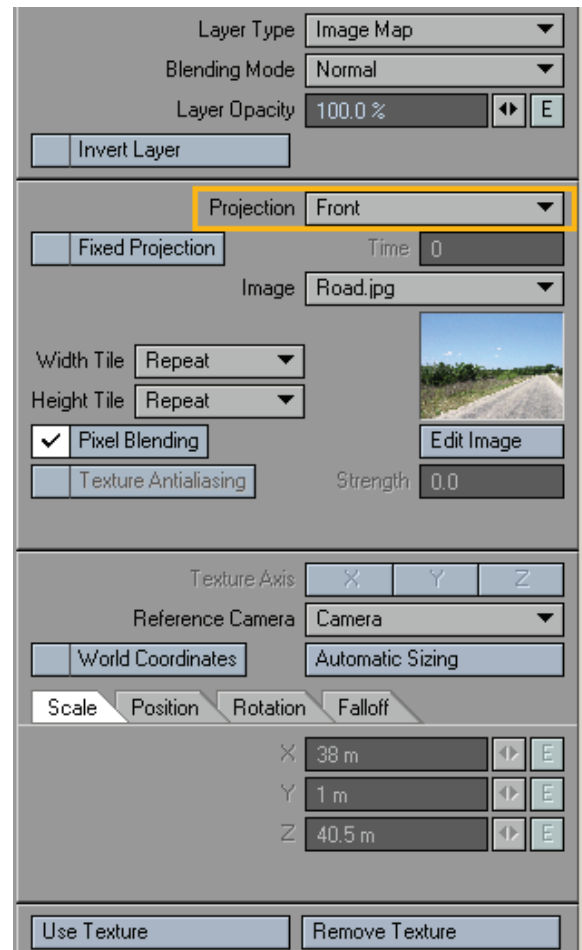
## Cubic Projection

**Cubic** projection is essentially the same as **Planar**, except that you cannot select a **Texture Axis**. **Cubic** projects the image from all three axes at the same time. The image is projected like **Planar**, except simultaneously along all three axes. Use **Cubic** when you wish to apply the same image to all sides of a rectangular shape, such as an image of tiling bricks wrapped around the sides of a building, or wallpaper on the walls of a room.

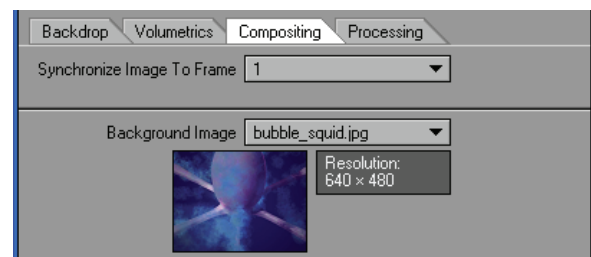


## Front Projection

The concept of **Front** projection is very simple and quite similar to a **chroma-key** effect. However, instead of applying an image where a color screen is, it replaces the selected surface(s) with a selected image.



In most cases, the image you select for a **Front Projection Map** is the same image you use for the **Background Image** on Layout's **Compositing Tab** of the **Effects Panel** (**Windows > Compositing Options**).



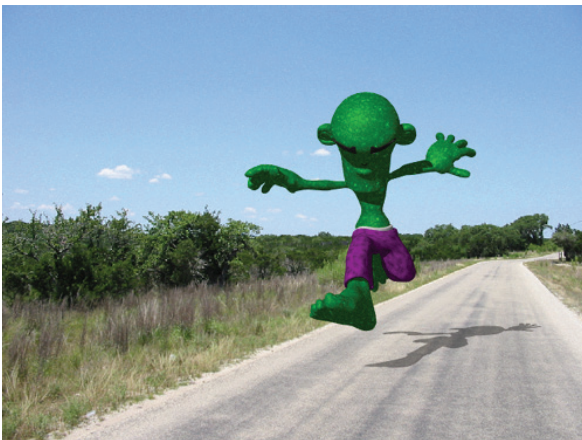
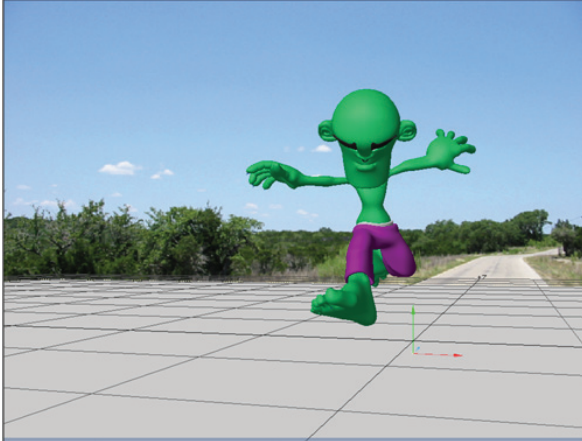
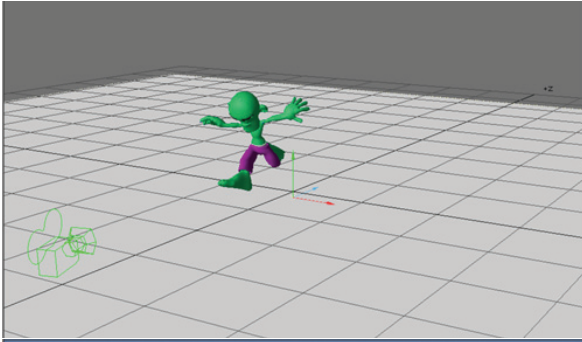
**Scale**, **Position**, and so on, are not relevant with **Front** projection. It is always the size (and frame/pixel aspect) it would be if loaded as a **Background Image**. As such, changing the **Resolution** or **Pixel Aspect Ratio** on the **Camera Properties Panel** will also affect the **Front** projection.

**Front** projection is used primarily for **comp** (compositing) work where you combine LightWave objects with a live-action background image or sequence. A common example occurs when you want a LightWave object to cast a shadow (believably) onto the image or **behind** a portion of the background image.





The image used in the surface and the background will **pin-register** the surface to the background, letting you go in front or behind. Your object then appears to interact with the environment. You can cast shadows or cause reflections from a regular 3D object onto the surface that is front projection mapped.



Front-Projection mapped Ground surface catches the character's shadow.

The ground object is just a flat box with **Front Projection Image Mapping** that uses the same image as the background. Its job is merely to catch the object's shadow.

Another example is to use an image of trees as your background image and fly a UFO between them so the UFO appears to go in front of some trees and behind others. All you need to do is model some rough shapes that match the trees you wish to fly behind (they could even be flat planes).

Another good example for **Front** projection is to create a flat plane and align it to an image of an ocean or a lake. **Front** projecting the water surface onto it lets you place an object beneath the water and push it through the surface. Submarines and sea creatures will appear to **break the surface** this way.

The hardest part of **Front** projection is aligning the objects, matching lighting, and getting the right camera angle. Using **Background Image** as the **Camera View Background** on Layout's **Display**

**Options Tab** of the **Preferences Panel** (**Edit > Display Options**) will allow you to do all that. You also must search for the right balance of **Luminosity** and **Diffuse** for the **Front** projection surface so that the object's true shape is not revealed by shading.



NOTE: Don't use the **Soft Filter** (**Camera Properties Panel**) with **Front** projection. It will soften only the image used on the object surfaces, not the background.

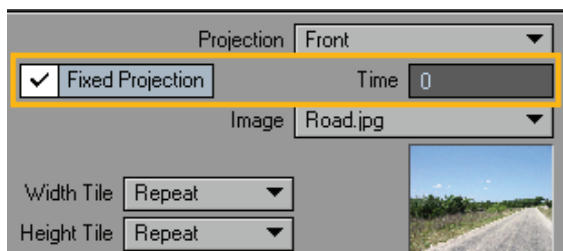


## Fixed Projection

**Front** projection surfaces will always look the same no matter where you move the object or which way you rotate it. The image does not normally stick to the surface. However, if you activate the **Fixed Projection** option (previously **Sticky Projection**), it **fixes** (i.e., locks) the projection from the camera's perspective at the specified **Time**.

The default unit of measure for **Time** depends on the **Frame Slider Label** setting on the **General Options Tab** of the **Preferences Panel** in Layout or the **Time Format** setting on the **Display Options Panel, Units Tab** in Modeler. You may specify the unit of measure by appending **f** for frames or **s** for seconds to the entered number (e.g., 22f for frame 22, 31s for 31 seconds). You may also enter SMPTE time code, like 00:00:01:16 for one second, frame 16. The entry is converted to the default unit of measure.

Use **Fixed Projection** to create parallax with two-dimensional images by using a technique called **Camera Mapping**. (Use the **Reference Camera** setting to select the camera, if you have multiple cameras in your scene.)



Essentially, you set the frame where the texture will be pin-registered to the background (like normal **Front Projection Mapping**). On all other frames, the texture is stretched to compensate for more or less of the texture being visible, which is caused by factors like the camera being moved.

For example, in a picture of some buildings, you could place the LightWave camera in the same place as the original camera in relation to some simple 3D buildings; then, you could project the picture onto the buildings and lock it at frame 0. You'll need a **doctored** background image/object — with **Fixed** projection — to reveal what's behind the buildings. If you move the camera forward, it will appear to fly into or around the buildings.



**HINT:** Use your paint program's rubber stamp function to erase areas where background **Fixed** projection surfaces will be revealed.

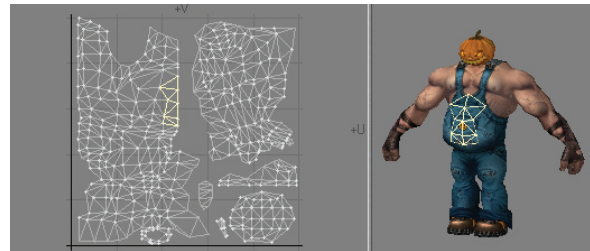


**NOTE:** Obviously, there are great limitations in getting three-dimensional data out of a two-dimensional picture. However, short slow-moving sequences can be quite realistic.

## UV Texture Maps

The **U** and **V** refer to **Texture Map** coordinates and are really not very different from the **XYZ** coordinates you are familiar with. In fact, **UV Mapping** is the process of setting up a relationship between the two dimensions of an image, **U** and **V**, with the three dimensions of an object surface, **XYZ**.

Once this relationship is set up, changing any parameter (i.e., **U**, **V**, **X**, **Y**, or **Z**) will also relatively change the appearance of the **Texture Mapping**. With **UV Mapping**, the object provides additional information for **Texture Mapping**, which can be different for any given point on the surface. The texture is more or less stuck to points on the surface using a relationship that you define.



## UVs and Projection

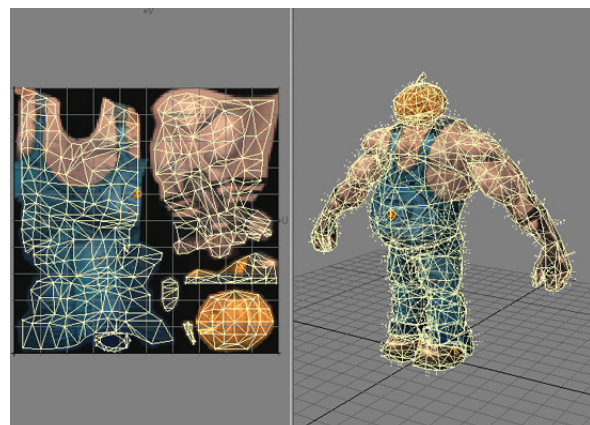
**UVs** have to come from somewhere. For existing polygonal models, the choices are limited to setting the **UV** coordinates for each point in the object manually, or applying some **projection**, which automatically generates the 2D **Texture** coordinates from the given 3D point positions. In LightWave, you can create **UVs** by using projections, which also happen to be the same as the standard projections for **Texture Mapping** (i.e., **Planar**, **Cylindrical**, and **Spherical**).

Usually, the projection for the **UV Map** is not suitable for the whole model. The projected **UV Map** must be tweaked — eyes and nostrils moved over the right parts of a face, or texture features matched to geometry features.



**NOTE:** Keep in mind that standard **Projection Mapping** is more accurate because it has some exact, continuous value over the entire surface. **UV Mapping**, on the other hand, is technically accurate only at small sample points. The surface for the large areas in between the sample points is interpolated. Adjusting the sample points so that the interpolated areas look right is difficult and the reason why **UVs** are more difficult to use.

For illustration purposes, let's say you had your texture image printed on a piece of very flexible rubber and wanted to fit it on a toy car made of wood. You could conform the rubber material to contours of the car by tacking it down with thumbtacks. That is more or less what **UV Mapping** does. However, it is slightly reversed: what you do is **tack** the **UV** points down onto the image.





## Layer Type: Procedural Texture

---

An alternative to using an image as a Texture is to use built-in mathematical computations called **Procedural Textures**. These serve the same purpose as **Image Maps**; however, since they are mathematically computed, they are applied seamlessly on a surface. Thus, you do not need to worry about a projection method.



All surfaces in the images above were created with procedurals.



NOTE: All **Procedural Textures** are available for use on each surface attribute.

---

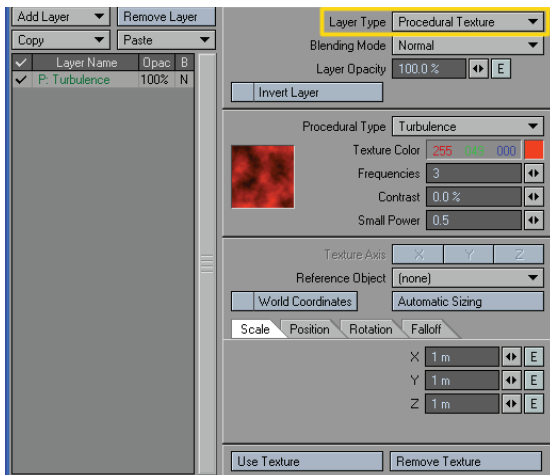






### To add a procedural texture layer:

Click on the **Add Layer** button and select **Procedural** from the pop-up menu. You can also change an existing layer's type by changing its **Layer Type** to **Procedural Texture**.



### Texture Color and Texture Value

Unless you are adding a texture to surface **Color**, you must specify a **Texture Value**, which is the value for the texture pattern at its most intense points. This can even be a negative value.

If you are adding a texture to surface **Color**, you do not define a value but a **Texture Color**. The color chosen here will be the color of the texture at its most intense points.

Unless the texture pattern fills all of the surface area, there will be transparent and semi-transparent areas, revealing underlying layers. In areas where the texture is below 100%, the texture will blend with underlying layers.

You can change the background color used in the thumbnail window by right-clicking on it—a **Color Selection** dialog will appear. You can also drag the texture around with your **LMB**.

### Texture Scale

Generally, you want your scale to be a fraction of the surface size to properly see the texture on the surface. Unless you know the exact dimension of your surface, it may be handy to click **Automatic Sizing** first and then modify the resulting **Scale** values.



**HINT:** Use LightWave's maths feature and append, say, **"/4"** to the automatic-sizing values. This divides the value by four. For example, adding **"/4"** to a value of **"24 m"** would look like **"24 m/4"** (don't enter quotes). When you move to the next field, the result is calculated automatically!

### Procedural Texture Settings

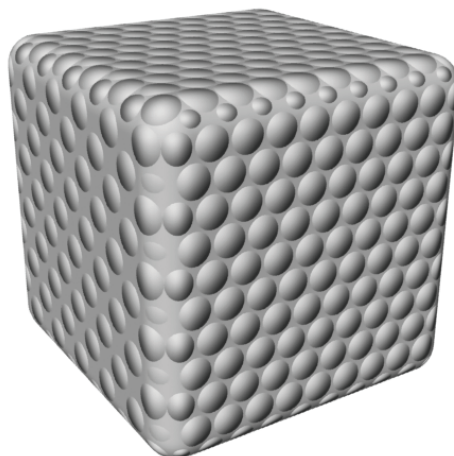
The following is a list of the **Procedural Textures**. Remember that although they are described in essentially surface color terms, they can also be used for **Diffuse**, **Specularity**, **Transparency**, and so on.

#### Brick



**Brick** produces an array of symmetrically spaced bricks. The **Texture Color** is the color of the mortar. **Mortar Thickness** determines the width of the mortar. At 1.0, no brick is visible. **Fuzzy Edge Width** makes the outer brick edges soft and faded. Since **Procedural Textures** are three-dimensional, you may need to tweak **Size** and **Position** values if a surface cuts through mortar and that is all you see on an edge.

#### Bump Array



**Bump Array** produces an array of symmetrically spaced circular dots that appear to have depth. You can use this texture to create a golf ball surface.

**Radius** sets the radius of each bump used in the bump array pattern. **Spacing** sets the spacing between each bump.

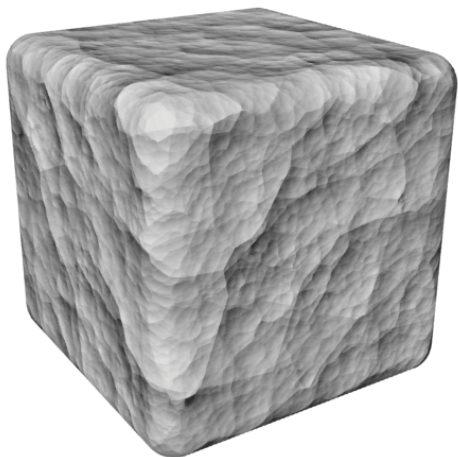


## Checkerboard



**Checkerboard** generates a tiled appearance on an object's surface, like that of a checkerboard or chessboard. Your **XYZ Scale** values set the size of each square in the pattern. For example, to see ten checkerboard squares on the surface of an object, divide its length by ten. A 50-meter cube would yield a **Scale** of 5 (for X, Y, and Z) to achieve ten squares.

## Crumple



**Crumple** is a very detailed texture that gives the appearance of a surface that was **crumpled** under pressure. You can use it to simulate crumpled paper, hammered metal, ice cubes, stucco, and even cauliflower.

**Frequencies** refers to the number of different scales of detail added to the pattern. A value of 1 makes a surface appear partially crumpled so that it has only a few large dents. A value of 5 makes a surface appear very crumpled so that it has many smaller dents.

**Small Power** affects the appearance of the large and small features added to a surface. The **Default** values create the appearance of a surface with large and small dents. A higher **Small Power** (1.0 or above) causes the smaller dents to be shaded with the same degree of intensity as the larger dents so that the surface becomes busier and the larger dents lose some of their distinction. A lower **Small Power** (under .50) makes for less distinction between the large and small features.

## Crust



**Crust** is one of the more complex textures. It produces raised circular splotches on a surface. This is good for barnacles, warts, or moon craters.

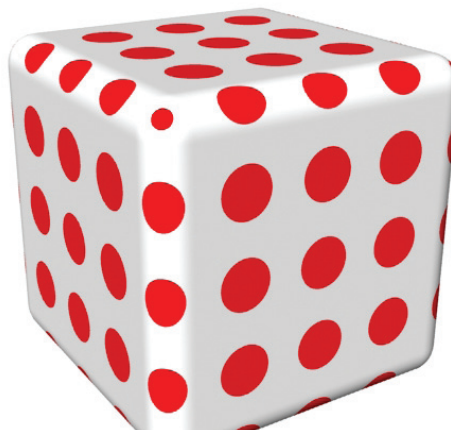
**Coverage** determines how much of the crusty pattern covers the surface. A low value (0.0 to 0.5) makes many small islands of crust across the surface, whereas if you use a high value (0.6 or higher), the crust pattern begins to cover the surface color underneath.

The patterns that display **Ledge Level** are shaded so that the surface appears to have raised edges. **Ledge Level** shifts the position of the **ledge** that defines the edges of the pattern further outward from the pattern itself, into the surface color. The values that make visible changes in these patterns are subtle, so change them in small increments until you understand their effects well (on the order of 0.1 at a time).

**Ledge Width** alters the apparent depth of the **valleys** formed by the ridges in the pattern. The values that make visible changes in these patterns are subtle, so change them in small increments until you understand their effects well (on the order of 0.1 at a time).

**Ledge Width** will affect the sharpness of the slope falling off of the discoloration pattern. The **Default** values create a ridge just along the outside of the pattern discoloration. Higher values move the ridge further outward from the pattern, lower values move it inward. The most useful values will fall between 0.3 and 1.0.

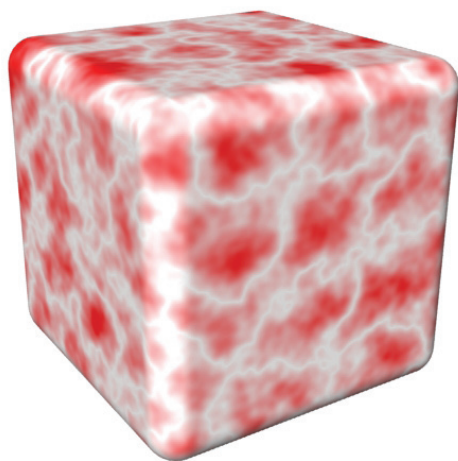
## Dots



**Dots** produces an array of evenly spaced dots. The dots can optionally have a soft edge. **Dot Diameter** determines the size of the dots in the dot pattern. At 1.0, the edges of dots will touch. **Fuzzy Edge Width** makes the outer dot edges soft and faded.



## FBM



**FBM** is another fractal pattern that uses the same settings as **Fractal Noise**.



**FUN FACT:** Fractional Brownian Motion (FBM) is named after botanist Robert Brown, who observed the random movement of pollen grains. Einstein later explained that the movement was due to surrounding molecules moving in random directions.

## Fractal Noise



**Fractal Noise** produces a random fractal pattern. It is undoubtedly the most commonly used texture, since it can quickly mask somewhat the computerised look of 3D imagery. Use this as a **Bump Map** for terrains and brushed metal (use an exaggerated value along the **grain** direction). As a **Transparency Map**, **Fractal Noise** can generate realistic-looking clouds. As a surface **Color** (or diffuse) **Map**, it can give a weathered look to surfaces, such as grass or dirt ground. You'll find numerous possibilities for this texture. (Also see the **Turbulence** texture.)

**Frequencies** affects the level of detail produced in the noise pattern. Increasing this level will increase the variation in the pattern. Values above 6 are not useful (the level of detail is so small that it may not be noticeable, and it will increase rendering time unnecessarily).

**Contrast** adjusts the blending of the texture. The higher the level (above 1.0), the greater the contrast, and the more pronounced the pattern. Values lower than 1.0 (between 0 and 1.0) produce a less stark, more softly blended pattern. **Small Power** refers to the amount of intensity applied to both large and small details. It changes the intensity of the smaller details. A higher **Small Power** (1.0 or above) causes the smaller dents to be shaded with the same degree of intensity as the larger dents so that the surface becomes busier and the larger dents lose some of their distinction. A lower **Small Power** (under .50) makes for less distinction between the large and small features.

Use very small **XYZ Scale** values in a **Bump Map** to add random pits to surfaces.

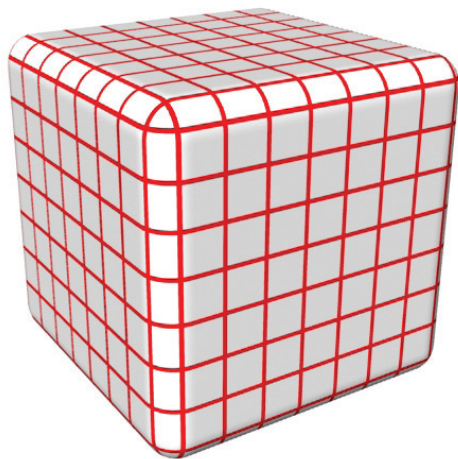


**NOTE:** The **Fractal Bumps** option in previous versions of LightWave — only available for **Bump Maps** — was really **Fractal Noise** with **Contrast** and **Small Power** set to .5.





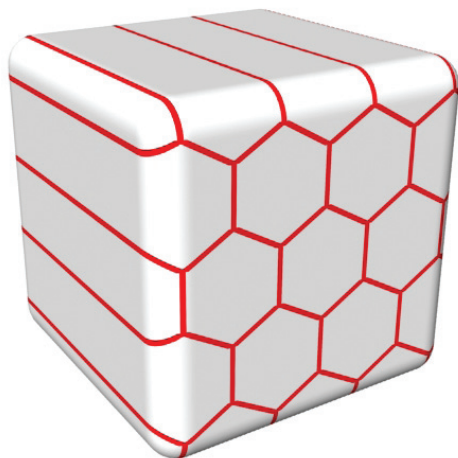
## Grid



**Grid** generates a grid across an object's surface. The grid is a three-dimensional grid. Therefore, lines project into all three dimensions. Often, you want a 2D grid superimposed on an object's surface, like graph paper. In such cases where you use the **Grid** texture and see unwanted portions of the grid, try expanding the size of the **Grid** texture along that axis (this expands it off the surface of the object). For example, if you map a grid onto a one-meter ball, the **Texture** scale of the Z axis can cause the appearance of ripples that break up the nice graph paper look of the X and Y **Texture** scale. Using a **Texture** scale of 0.25m, 0.25m, 1m will get the proper look.

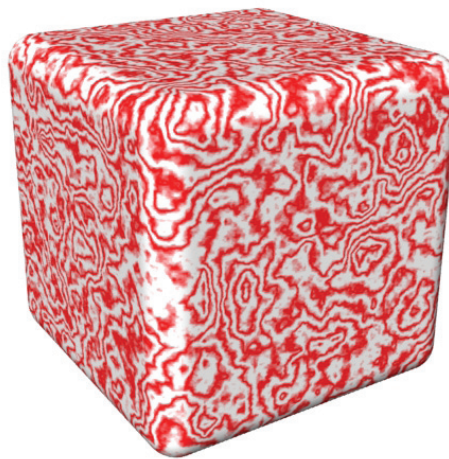
The **Line Thickness** value determines the thickness of the lines in the grid pattern.

## HoneyComb



**HoneyComb** produces a **hive** of activity in the form of a honeycomb. The **Texture Color** is the color of the lines. **Line Thickness** determines the width of the lines. **Fuzzy Edge Width** makes the outer dot edges soft and faded. Unlike Grid, Honeycomb is a two-dimensional texture where axis is important.

## Marble



**Marble** produces fractal-like patterns that imitate marble veins. The pattern is calculated in veins that wrap around a selected axis, just like rings wrap around the center of a tree.

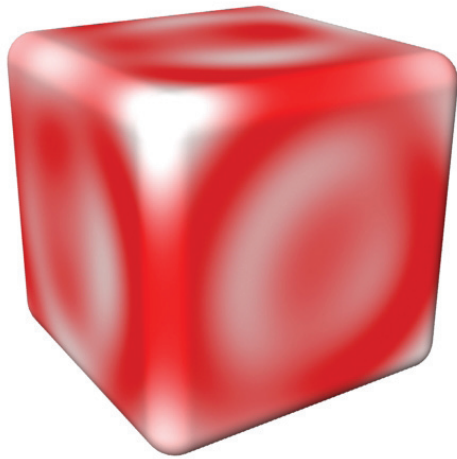
**Frequencies** sets the level of detail produced in the pattern. The higher the value, the better-looking the pattern becomes, but the longer it takes to render as well. (A value above 6 is not useful here, because the variance is microscopic in size and virtually invisible.) The **Turbulence** value should be a fraction (often one-half) of the **Vein Spacing** value. It determines how close a vein may come to a neighbouring vein. **Vein Spacing** sets the distance between veins in the marble pattern. **Vein Sharpness** sets the level of blending/contrast of the pattern. The lower the value, the more blending will occur. Higher values will produce very sharp, distinct veins.



HINT: First select a **Vein Spacing** and **Texture Axis** that you like, then set **Frequencies**, **Turbulence**, and **Scale**.



## Ripples



**Ripples** and **Ripples2** produce the appearance of fluid waves or ripples to a surface. Use small ripples to simulate water, or large ripples to put a wave in a flag.

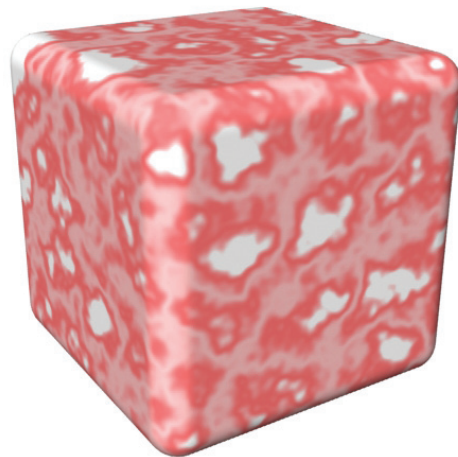
**Wave Sources** determines the number of ripple sources. The higher the value, the greater the number of areas rippling with waves. A value of 1 will create a single ripple pattern, like that of a solitary water droplet falling into a pond. Values higher than 16 are not recommended; they require longer rendering time and may not add to the appearance of the texture.

**Wavelength** controls the distance between the ripples. The lower the value, the closer the waves will appear to each other. **Wave Speed** sets the speed of the ripples.

### Looping Wave Ripples

In order to loop the movement of ripples throughout the course of the animation, use this formula to determine the proper **Wave Speed**: **Wavelength** divided by the number of frames over which the pattern should loop equals the **Wave Speed** (i.e., **Wavelength**/number of frames to loop = **Wave Speed**).

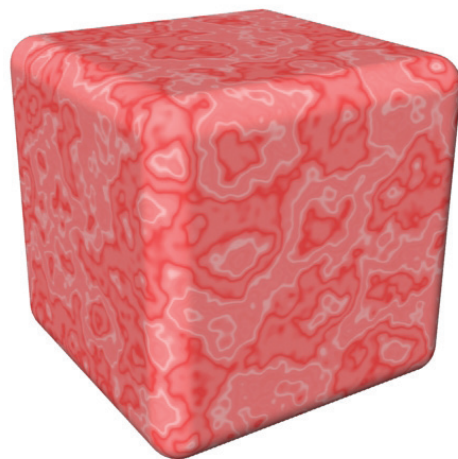
## Smoky 1, 2, and 3



Smoky 1



Smoky 2



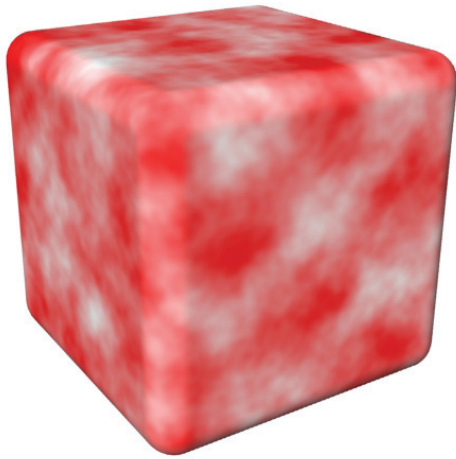
Smoky 3

The **Smoky** textures use the same settings as **Fractal Noise** with the addition of a **Turbulence** control, which lets you adjust the pattern disturbance.



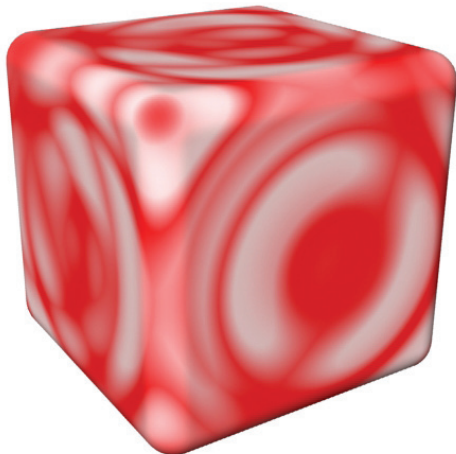


## Turbulence



**Turbulence** combines **Fractal Noise** layers, each at a different scale (or frequency). It produces an effect very similar to **Fractal Noise**, but will normally yield more eye-pleasing results. It also gives better control over the range of frequencies found in the noise, from smooth to detailed, grainy textures, due to the small scale noise that is added in. The settings are the same as those for **Fractal Noise**.

## Underwater



**Underwater** produces the rippling pattern effect of refracted light which you would see for instance on the bottom of a swimming pool. You can also use this texture to simulate nighttime sky effects such as the Aurora Borealis, changes in cloud patterns, or even electrical shocks.



NOTE: You can generate actual real world effects by using Layout's **Caustics** feature (**Lights > Properties > Global Illumination > Enable Caustics**), but rendering times will be significantly greater.

**Wave Sources** determines the number of ripple sources. The higher the value, the greater the number of areas rippling with waves. A value of 1 would create one ripple. Values higher than 16 are not useful. **Wavelength** controls the distance between the ripples. The lower the value, the closer the waves will appear to each other. **Wave Speed** sets the speed of the ripples. **Band Sharpness** sets the level of blending/contrast of the pattern. The lower the value, the more blending will occur. Higher values will produce very sharp, distinct bands.

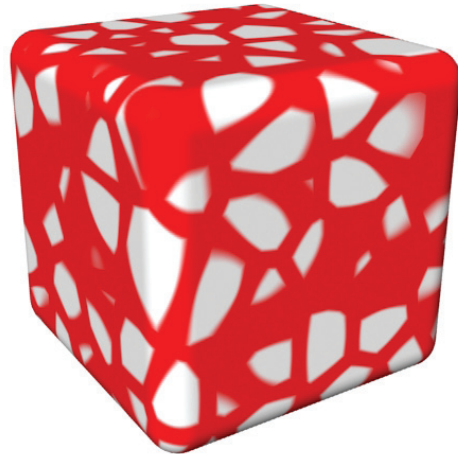
## Looping Wave Ripples

In order to loop the movement of ripples throughout the course of the animation, use this formula to determine the proper **Wave Speed**: **Wavelength** divided by the number of frames over which the pattern should loop equals the **Wave Speed** (i.e., **Wavelength/number of frames to loop = Wave Speed**).

## Value

The **Value** procedural lets you create a layer of uniform value or color. You can use this to composite layers with **Alpha Channels**.

## Veins



**Veins** produces a series of raised areas separated by canals or veins. This is great for cracked mud, leaded glass, stone walls, leaves, and so on.

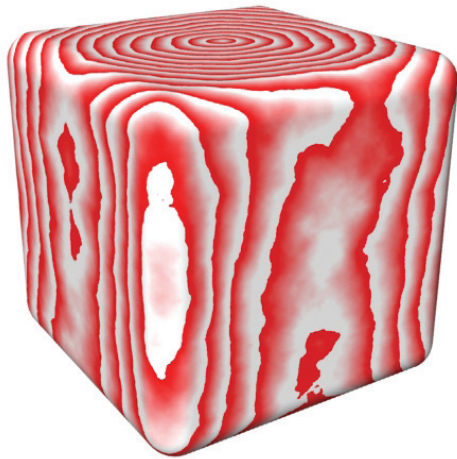
**Coverage** determines how much of the vein pattern covers the surface. A low value (0.0 to 0.4) applies the vein color only to veins themselves, whereas a high value (0.5 or higher) causes the vein color to fill in between the veins, overtaking the surface color underneath.

The patterns that display **Ledge Level** are shaded so that the surface appears to have raised edges. **Ledge Level** shifts the position of the **ledge** that defines the edges of the pattern further outward from the pattern itself, into the surface color. The values that make visible changes in these patterns are subtle, so change them in small increments until you understand their effects well (on the order of 0.1 at a time).

**Ledge Width** alters the apparent depth of the **valleys** formed by the ridges in the pattern. The values that make visible changes in these patterns are subtle, so change them in small increments until you understand their effects well (on the order of 0.1 at a time). **Ledge Width** affects the depth of the veined ridges. Values just above the default setting have the most visible effect. The default values create a ridge just along the outside of the pattern. Higher values shift the ridge further outward from the pattern, lower values move it inward. Most useful values fall between 0.2 and 1.0.



## Wood



**Wood** is similar to **Marble**, but produces a pattern imitating the rings in a piece of wood.

**Frequencies** sets the level of detail produced in the pattern. The higher the value, the better-looking the pattern becomes, but the longer it takes to render as well. (A value above 6 is not useful here, as the variance is microscopic in size and virtually invisible.) The **Turbulence** value should be a fraction (often one-half) of the **Ring Spacing** value. It determines how close a wood ring may come to a neighbouring ring. **Ring Spacing** sets the distance between rings in the pattern. **Ring Sharpness** sets the level of blending/contrast of the pattern. The lower the value, the more blending will occur. Higher values will produce very sharp, distinct rings.



HINT: First start by selecting a Ring Spacing and Texture Axis that you like, then set Frequencies, Turbulence, and Scale.

## Additional Procedural Textures

A number of additional **Procedural Textures** are available; these textures are based on the noise and fractal routines presented in the textbook **Texturing and Modeling: A Procedural Approach** by David Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin, and Steve Worley (Morgan Kaufmann Publishers, 2nd Ed., July 1998, ISBN 0122287304).

**Fractals** are the most commonly used tool in computer graphics for adding visual complexity to 3D surfaces. **Fractals** create this complexity by repeating a common underlying noise pattern at various scale sizes, and accumulating those scaled patterns into a final surface texture.

The following parameters are common to most of the following **Procedural Textures**:

**Increment** controls the fractal dimension of the texture. When set to zero, the fractal function looks like white noise. As the value gets larger, the fractal will become smoother.

**Lacunarity** is the amount of change in the frequency of noise that occurs between each successive iteration of the fractal calculation. As this parameter increases, the average size of the gaps between the scaled patterns will also increase.

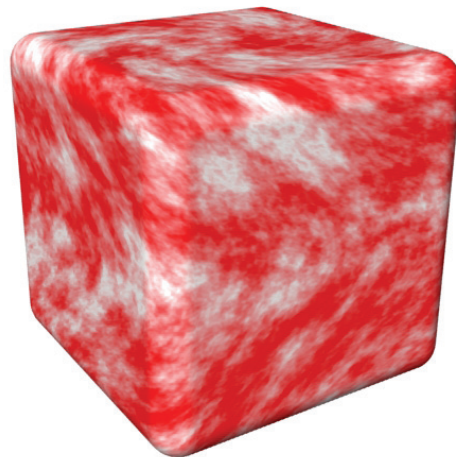
**Octaves** are the number of times the texture is scaled down to a smaller pattern, and added back again to the larger pattern. Larger values add more small details to the final texture, but also increase rendering times.

**Offset** exists in all the multi-fractal procedurals (where the fractal dimension varies throughout the texture). When set to zero, the fractal dimension will change greatly throughout the texture. This causes the roughness of the texture to vary greatly across the surface of the texture. Larger values cause the roughness (or smoothness) to be more consistent across the entire surface of the texture.

**Threshold** specifies a **Threshold** value that is used to determine whether the texture should be displayed or not. If the **Procedural Texture** value is higher than **Threshold**, that value modifies the surface. If the **Procedural Texture** value is lower than **Threshold**, that value does not alter the existing surface attribute at all.

**Noise Type** uses **Perlin Noise** as the most common and fastest noise function available in these procedurals. The other options (**Value**, **Gradient**, **Value-Gradient**, **Lattice Convolution** and **Sparse Convolution**) are different implementations of noise written by Darwyn Peachey and described in chapter 2 of **Texturing and Modeling: A Procedural Approach**, referenced above. While these implementations may provide better quality noise, they are definitely slower than **Perlin Noise**.

## Coriolis



**Coriolis** is a texture used to simulate the shearing effect of the atmospheric flow on earth caused by the earth spinning faster at the equator, and slower towards the north and south poles.

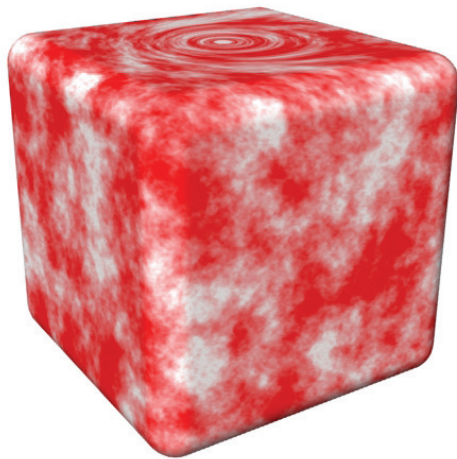
**Coriolis Scale** scales the value calculated by the **Coriolis** texture. Its effect is to vary the contrast between the **clouds** and the underlying surface. Smaller values create less contrast, and larger values display a high contrast between the clouds and the surface.

**Coriolis Twist** is the amount of twist or rotation of the clouds from the poles to the equator.

**Coriolis Offset** is added to the value calculated by the **Coriolis** texture. Larger values create denser clouds and smaller values result in fewer, thinner clouds.



## Cyclone



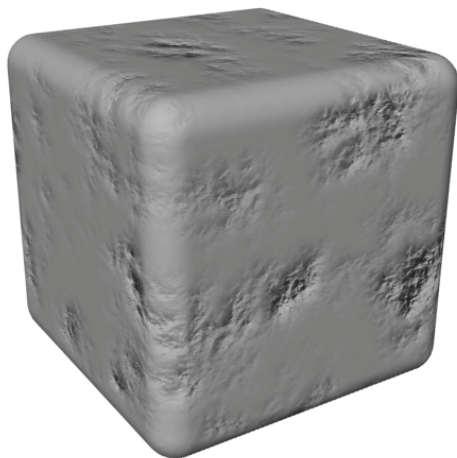
**Cyclone** is a **Turbulence** texture with a single vortex, used to simulate cyclones and hurricanes.

**Cyclone Radius** is the maximum radius of the cyclone. Outside of this radius, the clouds will not appear twisted.

**Cyclone Twist** is the amount of twist or rotation of the **clouds** within the **Cyclone Radius**.

**Cyclone Offset** is added to the value calculated by the **Cyclone** texture. Larger values create denser clouds and smaller values result in fewer, thinner clouds.

## Dented

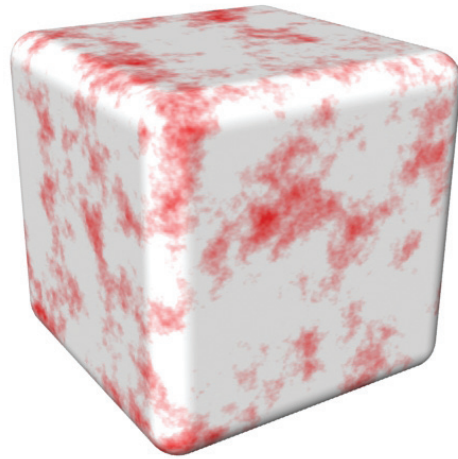


Dented is a turbulent noise function that creates crumpled dent patterns. **Scale** adjusts the magnitude of the texture output.

**Power** is the fractal dimension of the **Dent** texture. A value of 1.0 looks like crumpled paper. 3.0 makes smaller, isolated dents on the surface.

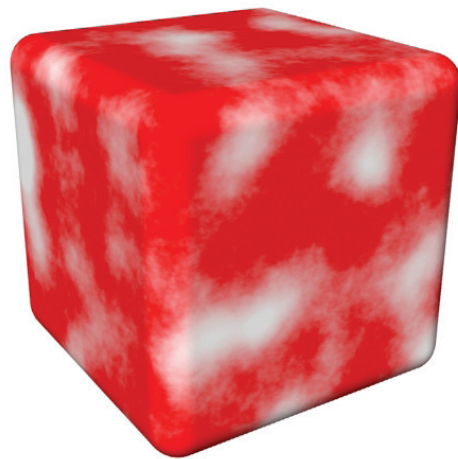
**Frequency** is the frequency of the dents, affecting the detail of the pattern.

## FBM Noise



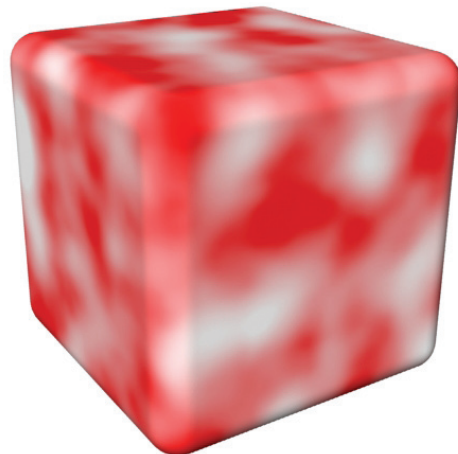
**FBM Noise** is a typical homogeneous fractal noise function since the fractal dimension does not vary.

## Hetero Terrain



**Hetero Terrain** is a multi-fractal texture that is smoother at lower elevations and progressively gets rougher as the altitude increases.

## Hybrid Multi-fractal

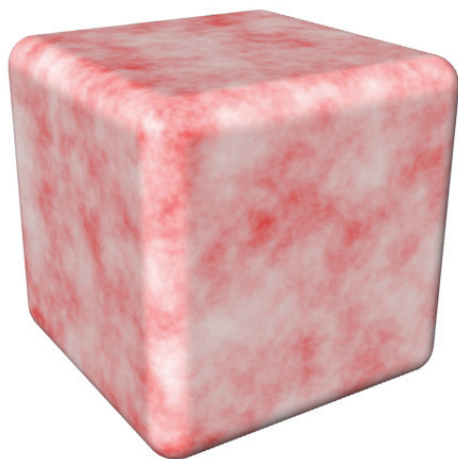


**Hybrid Multi-fractal** is another multi-fractal texture that smooths the **valleys** of the texture at all altitudes, not just at lower elevations.



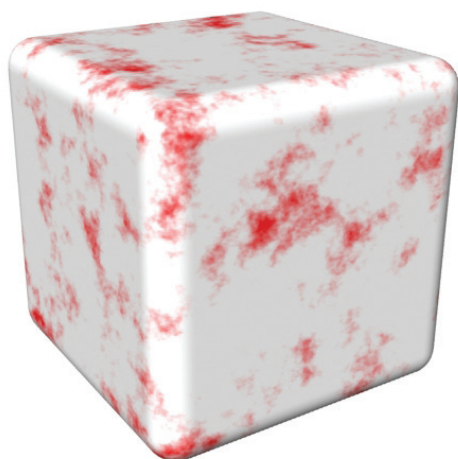


## Multi-fractal



A **Multi-fractal** is a function whose fractal dimension varies depending on the location of the point being shaded or displaced. This is similar to **FBM**, except that it uses multiplication in its inner loop computation rather than addition.

## Puffy Clouds



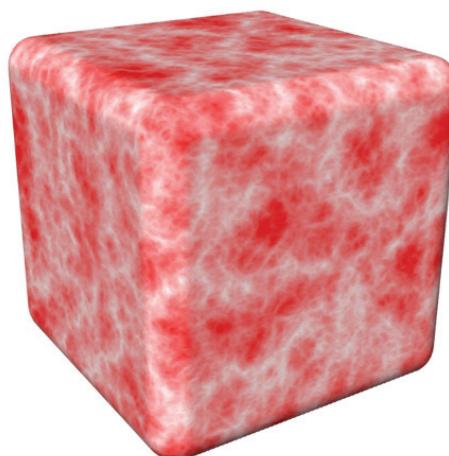
**Puffy Clouds** is a **thresholded FBM** noise function that creates soft, puffy cloud patterns.

## Ridged Multi-fractal



**Ridged Multi-fractal** is a hybrid multi-fractal texture that uses a **Threshold** value to create ridges in the terrain.

## Turbulent Noise

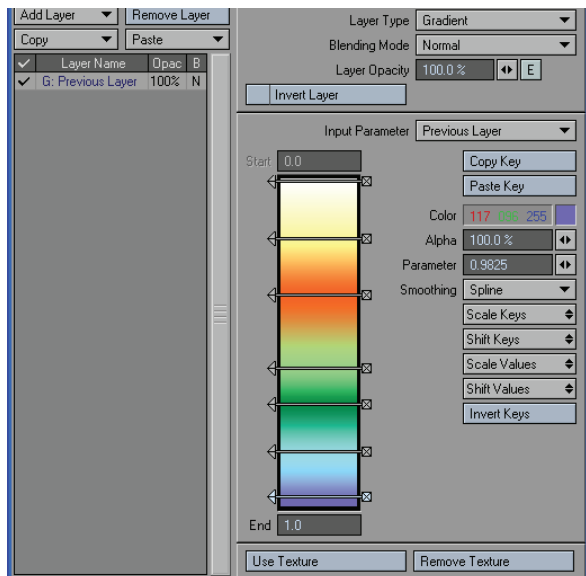


**Turbulent Noise** (formerly **TurbNoise**) is a modified **FBM** texture that uses an absolute value function, adding a turbulent effect to the texture.



## Layer Type: Gradient

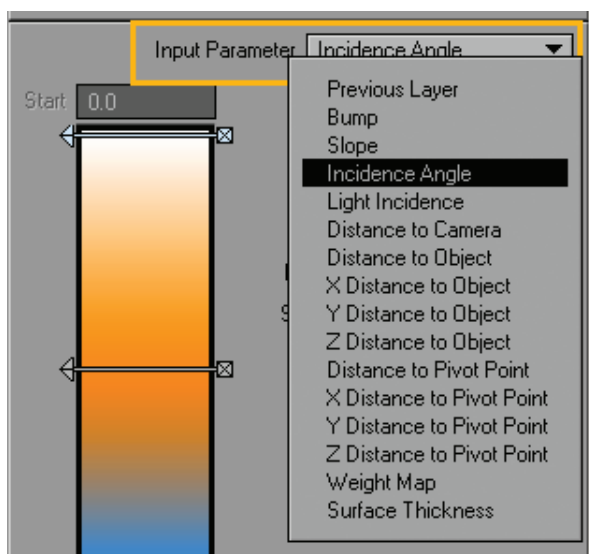
While **Envelopes** let you vary a setting based on time, a **Gradient** lets you vary a setting (**Color**, **Luminosity**, etc.) based on features like height of surface bumps, or distance to another object. So, for example, with an **Envelope**, you can make the **Diffuse** level 20% at frame 23. With a **Gradient**, you can make the **Diffuse** level 20% if a surface is 3.23 meters from the Cow object or the slope angle of the surface is 26.2 degrees.



The **Input Parameter** determines what feature the setting varies over. Think of **Gradients** as **filters** for the selected **Input Parameter**. For example, if the surface is bumpy, colors can be different based on the height of the surface being colored if the **Bump** option is used. You can even use the preceding texture layer.

**Gradients** use **gradient ramps** (the colored bar) to depict the value change. You define different values along the bar. LightWave automatically determines the in-between values. Essentially, the different colors along the gradient correspond to texture color/values for the related **Input Parameter**. The gradient is just a colorful (and clever) way to create a graph.

The **Input Parameter** defines what item will dynamically control the parameter.



**Previous Layer** uses the brightness data from the immediately underlying layer.

**Bump** uses the height of a surface using Layout's **Default Unit** on the **General Options Tab** of the **Preferences Panel**.

**Slope** changes the parameter based on the angle (in degrees) of the surface relative to the ground (flat on Y axis).

**Incidence Angle** uses the angle (in degrees) of the surface relative to the Camera.

**Light Incidence** is similar to **Incidence Angle**, but the angle is relative to the selected light. Use the **Light** pop-up menu that appears.

**Local Density (HyperVoxels only)** uses the density of the **HyperVoxel**.

**Distance to...** settings change the parameter independently based on the distance from the surface to the selected camera or object. The **X**, **Y**, and **Z Distance** settings measure the distance only along the respective axis. If you select the distance to an object, an **Object Selection** pop-up menu will appear.

**Weight Map** uses the **Weight Map** selected on the **Weight Map** pop-up menu.

**Surface Thickness** is a method of faking sub-surface scattering. **IMPORTANT:** The surface must be set to double-sided and **Trace Transparency** needs to be activated.

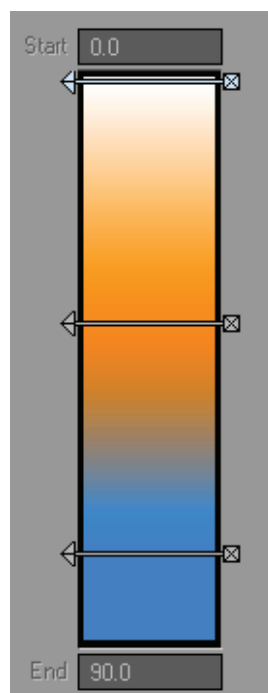


NOTE: Other **Input Parameter** options may be available depending on what feature the **Gradient** texture controls.



## The Gradient Bar

Once you have determined your **Input Parameter**, you need to set the value of the new texture color/value. The **Gradient Bar** allows you topographically set your **Key** values. (A **Key** is characterised by three numbers: its **Value**, **Parameter**, and **Alpha**.) As with **Envelopes**, you can create, edit, and delete keys. The **Keys** change the color of the **Gradient** and ultimately the **Parameter** values.



If the parameter is a color, the bar is in color and color transitions appear from one key to the next. If the parameter relates to numeric values, the bar appears in greyscale. The lowest set **Key** value is black and the highest is white.

The **Start** and **End** fields, at the top and bottom of the **Gradient Bar**, respectively, set the beginning and end of the visible area of the gradient. Whether or not these values are editable, depends on the **Input Parameter** you selected; however, their default values should work for most situations. Also, you can set keys outside this range.

### To add a key:

**Step 1:** Click on the **Gradient Bar** and hold down your LMB. A key marker that looks like an arrow will appear.

If you continue to hold the mouse button down, you can drag to refine the position of your key.

The **Parameter** field will change as you drag your mouse. Note that this value is not limited to whole numbers.



NOTE: To prevent accidentally adding a new key, you can drag the key using the arrowhead.

**Step 2:** Release the mouse button to create the key.

### To delete an existing key:

Click on the end of the arrow.



### The Current Key

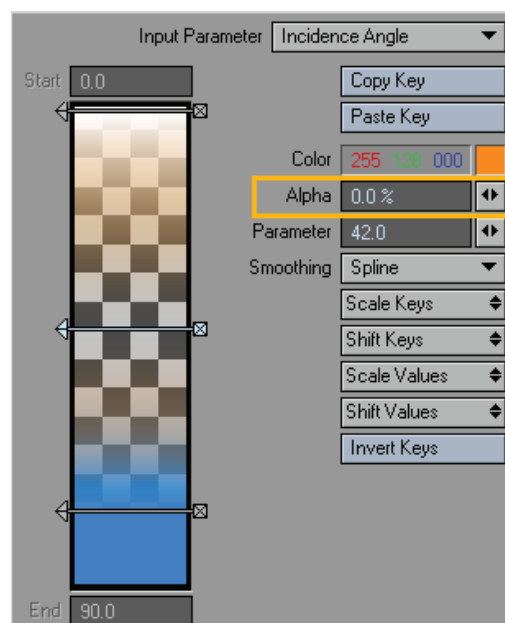
The **Value/Color**, **Alpha**, and **Parameter** fields show the corresponding settings for the **current key**. This is the key that is highlighted in the **Gradient Bar**. Click on a key's arrowhead to select it as the current key. When the panel is active, you can also use your **Up** and **Down Cursor keys** to select the preceding or next key as the current key. (Make sure that you do not also have an input field active.)



NOTE: When you edit a surface Color, the **value** is a color selector. All other types of textures have a numeric input field for value.

The **Parameter** setting determines the current key's numerical position on the bar. Note that this setting can have decimals. The **Value/Color** is the texture value or color for the current key.

The **Alpha** setting is the amount of opacity. Higher values are more opaque: 100% is fully opaque and 0% is fully transparent. You can see a background checkerboard when you use **Alpha** settings below 100%.





## Changing Key Values

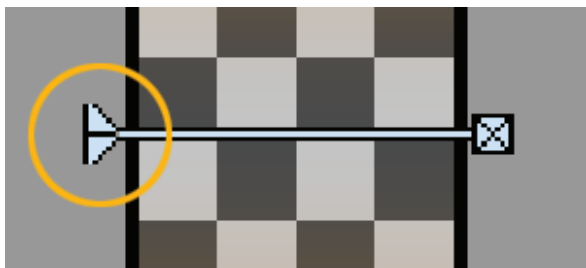
You can change the current key's **Color** or **Value**, as the case may be, by adjusting that setting. You see the **Gradient Bar** change as you adjust the setting.

### To prevent a key from being changed:

Right-click on the arrowhead. It will invert, indicating the key is fixed and cannot be altered.



Unlocked



Locked

## Changing Key Positions

You can move a key — but not past a neighbouring key — by dragging the key's arrowhead with your mouse.



NOTE: Although you can also drag the middle of the arrow, it's not recommended because you can accidentally add a new key.

You can also numerically move a key by entering a value (including a decimal if needed) into the **Parameter** field. Note that you still cannot move the key past a neighbouring key.

## Smoothing Between Keys

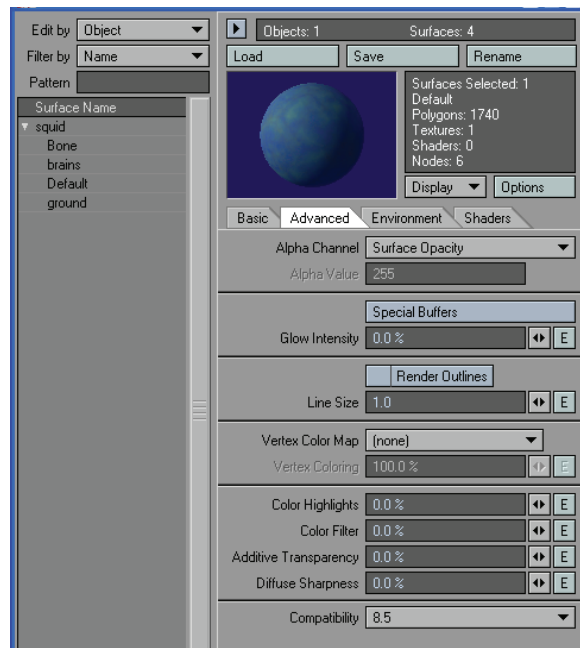
By default, the values between a key and the preceding key (next one above it on bar) are transitions in a **Linear** fashion. Setting the **Smoothing** pop-up menu to **Spline** will apply an **ease-in/ease-out** transition to the preceding key. The **Step** setting will hold the preceding key value until the next key.

## Scale and Shift

You can scale and shift the key **positions** by dragging up and down on the **Scale Keys** and **Shift Keys** buttons. Similarly, you can scale and shift the key **values** by dragging on the **Scale Values** and **Shift Values** buttons. **Invert Keys** reverses the values of the keys. This will not affect the position of keys on the bar, however.

## Advanced Tab

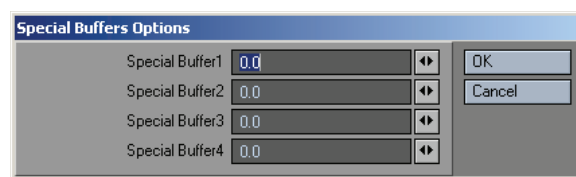
The **Alpha Channel** option affects the contents of LightWave's **Alpha Buffer**. Thus, the alpha image is saved when **Saving Alpha Images** is active on the **Render Options Panel (Rendering > Render Options)**. This option is applied on a surface-by-surface basis; each surface can have a different setting.



**Surface Opacity**, the default, uses the opacity of a surface to create the corresponding alpha area. Thus, if a surface has some level of transparency, the alpha channel information will be somewhere between 0 and 255. If you select **Unaffected by Surface**, the surface has no effect on the alpha image. When you select **Constant Value**, the **Alpha Value** field can be set to a specific number from 0 to 255. **Shadow Density** uses the density of shadows in creating the alpha image.

The **Special Buffers** function works in conjunction with certain image or pixel filters and lets you make surface-by-surface adjustments. Essentially, the settings are arbitrary values associated with a surface and can be used any way the filter wants. For example, the **Corona** image filter can use special buffers for its **Threshold Mask** setting.

When you click the **Special Buffers** button, a dialog will appear with four input fields. The values you enter depend on what the filter seeks.



NOTE: Use values from 0 to 1 for older filters requiring 0 to 255 integer values.

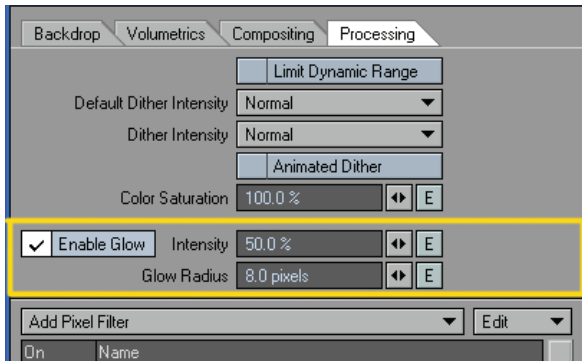
The **Special Buffer 1** field relates to the first filter in the image or pixel filter list on the **Processing Tab** of the **Effects Panel**. **Special Buffer 2** relates to the second image filter, and so on. The **Special Buffer 4** field works on the fourth filter and all that follows it.

Adding a **Glow Intensity** amount will render a glow around the selected surface. The overall amount and intensity of the glow are determined by the settings entered for (**glow**) **Intensity** and **Glow Radius** on the **Processing Tab** of the **Effects Panel (Window > Image Processing)**. The glow will appear in the color of the surface, even if it is texture mapped.



## Compatibility Menu

Scenes created before LightWave v9.0 had Surface Editor features which did not work as intended. The features have been updated. The Compatibility menu allows you to use older scenes created before LightWave v9.2 and match those scenes rendered in older versions of LightWave. The Compatibility menu is found in the Advanced Tab of the Surface Editor.



**NOTE:** **Enable Glow** must be active on the **Effects Panel** in order for LightWave to calculate glows around surfaces.

**Render Outlines** renders the outline of a polygon rather than its face. All of the polygons belonging to the current surface will render with their edges appearing in the surface color. If you apply an **Image** or **Texture Map** to the surface, the outlines will show portions of this mapping. Use this option to surface open frameworks or the gaps around hatches. **Line Size** is outline line size in pixels. Lines are drawn with the current surface color.

The **Vertex Color Map** pop-up sets the **Vertex Color Map** you wish to use. This feature colors the object vertices.

**Color Highlights** causes a surface's own color to **show through** a specular highlight. Normally, a specular highlight shows in the color of the light striking the object. For example, a white light illuminating a red ball leaves a white specular highlight on the surface of the ball. But with **Color Highlights**, the highlight appears red. A color highlight blends the object's surface color with the color of the light source, which causes a more metallic appearance. **Reflection maps** are also tinted by the surface color.

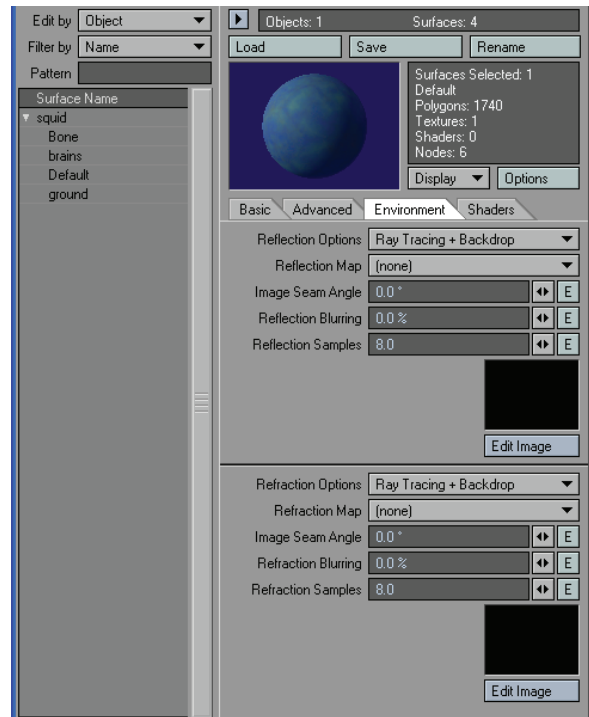
**Color Filter** is available when a surface has some degree of transparency. It allows the surface color of an object to tint all of the objects seen through it. A good example is a green wine bottle. Light shining through the bottle tints all objects seen behind it with green.

With **Additive Transparency**, the color of an object is added to the colors of the objects visible behind it. (Obviously, the surface must be somewhat transparent.) The surface therefore appears brighter in color and intensity, as if it was overexposed. The brighter color will tend ultimately toward white (255, 255, 255), because this is the brightest it can reach. Use this control to enhance the glow of rocket flames, jet exhausts, and the like.

**Diffuse Sharpness** causes a more severe **shadow cutoff** zone on objects rather than the gradual falloff of light. It enhances the appearance of contrast between the lit and unlit sides of the object. **Diffuse Sharpness** is usually used for very large-scale objects, like planets.

## Environment Tab

**Reflection Options** determines what you want reflected onto the surface. You can choose between different combinations of the backdrop (Background Image), actual ray tracing, and an **Image Map**. **Spherical Reflection Map** causes the reflective surface to reflect the image selected as the **Reflection Map**. This reflection is a spherical image that is mapped onto the inside of an all-encompassing sphere that surrounds the scene at an infinite distance.



**Image Seam Angle** determines where LightWave places the **seam** of the reflected image.

The 3D universe is considered a mathematical sphere, with all rendering occurring inside this sphere. LightWave uses the **Reflection Map** image to wallpaper the inside surface. In this manner, the image can be reflected off the different surfaces on objects within the scene, no matter which angle they face.

Unless the image you choose is seamless (i.e., it has **matched edges**), a visible seam appears where its edges meet. You may or may not see this seam in the reflection, depending on where you point the camera. If the seam is visible, you can adjust where it falls by changing the **Image Seam Angle**. This setting corresponds with **Heading** in Layout — as if you were standing at the center of the **Layout Grid** (0, 0, 0), and looking straight ahead into the positive Z axis.

The **Reflection Blurring** value allows you to add soft ray-traced reflections. The value controls the spread of the semi-random reflection rays. This is a sensitive setting and small values are recommended.

**Refraction Options** have the same available settings but affect what is refracted through the surface. Note that if you select **Spherical Refraction Map**, you will not see any objects through transparent surfaces. If you select **Ray Tracing + Spherical Map**, you should see other objects being refracted, with the **Spherical Map** behind them. The rays will bend and go off until they either hit an object or the map.



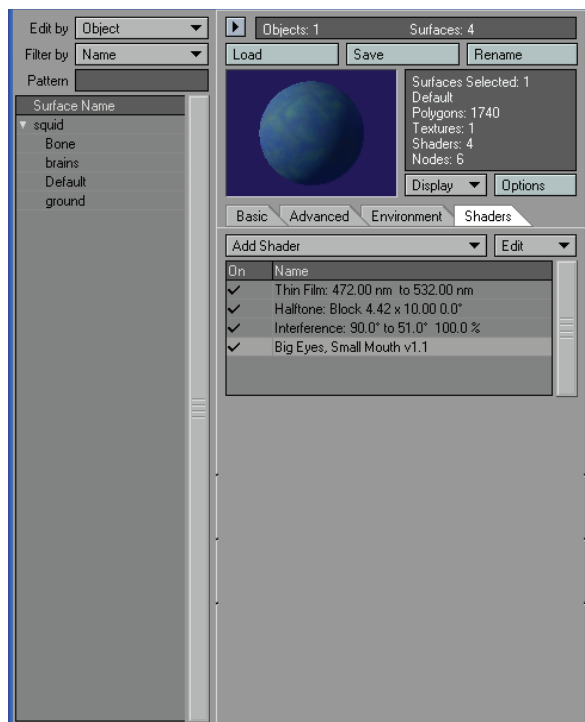
**NOTE:** If you use **Ray Tracing** options, you must activate the **Ray Trace Reflection** and/or **Ray Trace Refraction** options on the **Render Globals Panel** (**Render > Render Options**). These will also take much longer to render.





## Shaders Tab

LightWave lets you use surface shaders to change surface attributes. Here you can add, remove, and reorder shaders, as well as change their settings. To add the shader, select it from the **Add Shader** pop-up menu. To access its settings, double-click on the shader in the list. Depending on the shader, the options, if any, will either appear in their own dialog or below the shader list.



NOTE: If a shader does not appear in the pop-up menu and you think it should, you may need to add the plugin.

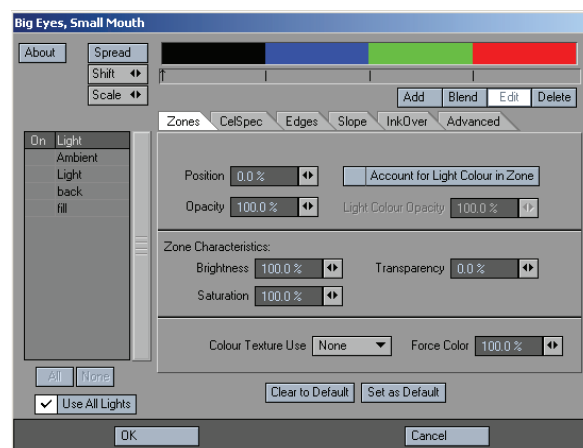
## AH\_CelShader

This is LightWave's original **Cel Shader**. **Cel Shader** has four color zones.



## BESM

**BESM** or Big Eyes Small Mouth is a cartoon shader. This expanded control lets you turn a surface into the look of a cel-shaded animation, like a traditional cartoon.



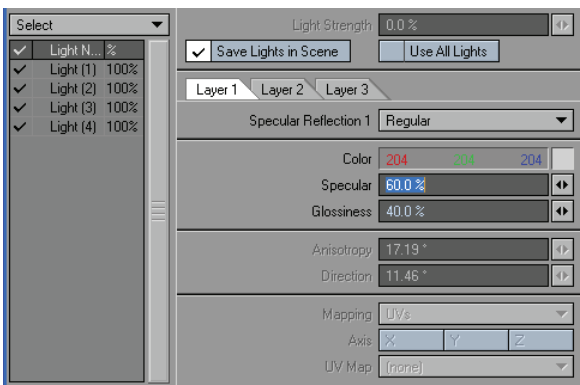


## BRDF

**BRDF** (Bi-directional Reflectance Distribution Function) is really a multi-function shader. First, by un-checking the light in the list, you can totally exclude specified lights from affecting the surface. .

**BRDF**'s second function is to let you stack up to three specular highlights with different settings. Multiple specular layers are important for surfaces like the car paint on a shiny red 1996 Mazda MX-5 named **Seiko**, where the first layer of paint is the color red and has a less glossy specular highlight and the top layer of paint is actually a sealer that is clear and high gloss. This will allow you to have a low-gloss colored specular under a high-gloss white specular.

Real world surfaces like machined metals reflect light unevenly, yielding what is often called a brushed-metal look. This is called **Anisotropic Distortion**, and it's the shader's third function. Compared to a smooth surface, these surfaces will have a softer and broader specular highlight.



On the left, you select which lights to affect. On the rest of the panel you set up multiple specular highlights, and make the shape of the highlight different by applying **Anisotropic Distortion**. The **BRDF Specular** settings work with the normal **Specular** surface attribute, so you must have some **Specular** on the surface to see any change.

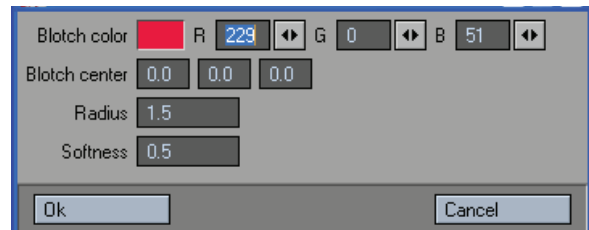
For **Anisotropic Shading**, you must know two directions lying in the surface. The specular reflection will be stronger along one direction. An example is a surface that has long highlights along fine grooves and short ones across them. With the **Anisotropy** and **AnisotropicII** reflection types, you can set **Alpha** and **Beta** settings, which essentially determine two different angles of disturbance. These effectively add tiny grooves to the surface that make the specular highlight imperfect — like a **micro bump**.

With **AnisotropicII** uniquely, you can specify a **UV Map** to define these directions. The **Alpha** parameter is essentially an angle for groove sharpness and blends the plain reflection with the **Anisotropic Reflection** — 90 degrees is 0% anisotropic. **Beta** is the angle between the the U direction and the maximum reflection, and so it rotates the grooves in relation to the **UVs**.

There are many models for **Anisotropy** and two are featured in this shader. Basically, these differ in the pitch and bank variance on a surface of the grooves.

## Blotch

This shader adds blotchy color to a surface. Choose the **RGB** value, **Center** of the blotch, **Radius**, and **Softness**. This LScript is documented in the LScript documentation as a **Shader Object Agents** example.

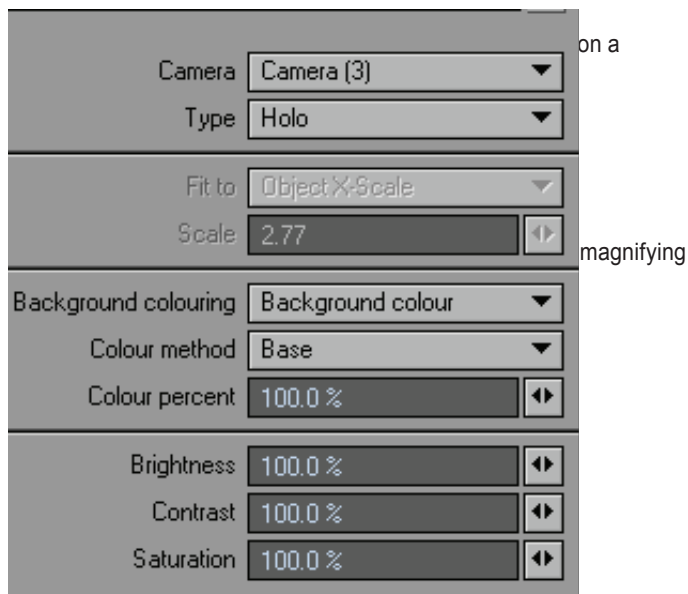


## Bumprgb

This shader converts surface normals to RGB levels which is useful for baking **Bump Maps** for dot3 real-time bump map pixel shading.



## CCTV



### CCTV usage

When CCTV is used to shade a point on a surface, it traces a ray from a given camera (which need not be the current render camera) into the scene. We'll call this camera the CCTV camera. The exact starting position and direction of the ray relative to the CCTV camera is determined by the settings for the CCTV shader.

The CCTV settings should be set in Layout. CCTV may be added and set up in Modeler, but as there are no cameras in Modeler the CCTV camera can only be set in Layout.

Once added to a surface, parameters specifying the CCTV camera to use, how to map the CCTV view onto the surface, and how to manipulate the color are given.

### Camera

The Camera settings has a drop-down list of all the cameras in the scene. Select the camera which you wish to become the CCTV camera. Note that only camera objects are listed. When set to none, CCTV is effectively disabled.

### Type

Sets the CCTV type to use. Select one of:

- TV
- Teleport
- Holo

The following two options are only available when the type is set to TV.

### Fit to

- None
- Object Y-Scale
- Object X-Scale

Scales the CCTV image to fix the scale of the given object axis. If set

to None, the scaling is determined only by the Scale setting.

### Scale

Sets a manual scale factor in conjunction with Fit to. If "Fit to" is set to Object Y- or X-scale, this scale multiplies the scaling determined by that setting.

### TV

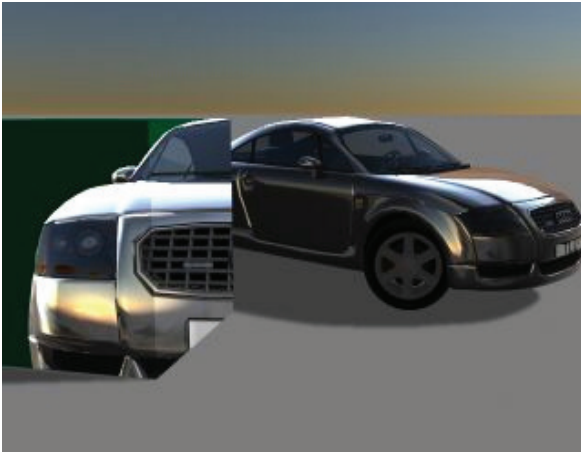


The TV type paints the view from the CCTV camera onto the XY plane of the object to which the surface is applied.

For the TV type the mapping from a spot on the surface to a CCTV camera ray is as follows. The X and Y coordinates of the spot are computed relative to the object's (possibly scaled) coordinate system. These coordinates are then reinterpreted to mean a point on the CCTV camera's image plane. A ray is then shot from the CCTV camera through that point on the CCTV camera image plane.



## Teleport



With Teleport, the ray from the render camera to point on the surface being shaded is “teleported” to come from the CCTV camera. The effect is as if the view from the CCTV camera is projected through the render camera.

Teleport is done by finding the direction of the ray from the spot being shaded on the surface to the render camera, relative to the render camera coordinate system. The ray from the CCTV camera will then be set to have the same direction relative to the CCTV camera.

## Holo



Using the Holo type causes the scene around the CCTV camera to appear like a hologram on the surface. This could be used for instancing.

The Holo type figures out the direction of the ray hitting the surface and the position of the spot being hit, relative to the object's coordinate system. This direction and position is then taken to be a direction and position relative to the CCTV camera. It is from this position and direction that a ray is shot into the scene.

## CCTV scaling and fitting

CCTV is sensitive to the scale of the object being shaded. Double the size of the object by a Size or Stretch command in Layout, and the image that CCTV draws on the surface will double in size as well.

For the TV type it is additionally possible to set whether to scale the image to be as large as the object's X or Y scale.

## CCTV coloring

CCTV can apply color to the surface in two ways. First, it can make the new color the base color of the surface. This is the color of the surface before lighting is applied. The second way is to replace the color after lighting. Lighting does not affect this color. It can be used to give the impression of an image being emitted like from a television, as opposed to looking like a photo stuck on the surface.

Televisions usually have brightness, contrast, and saturation controls. So does CCTV. To make a black & white TV, just turn down the saturation.



## Color application

What to do with the color that the CCTV camera applies to the surface is controlled by these settings.

### Background coloring

This affects what happens when a ray from the CCTV camera does not hit anything in the scene. When set to Surface color, the surface color remains unchanged from what you'd get if CCTV was turned off. When Transparent background is used, the surface is set to be completely transparent. This is particularly useful in combination with Holo. The Background color setting uses the color that the ray from the CCTV camera sees.

### Color method

When set to Base, CCTV will affect the base color of the surface. This is the color before other shadings and lighting. When set to Replace, the color set by CCTV is the final color of the surface.

### Color percent

Specifies how much CCTV contributes to the surface color.

### Color controls

CCTV can alter the colors it applies to the surface using controls similar to a regular television set.

#### Brightness

Controls the brightness. 100% is normal, less moves the color towards black, more towards white.

#### Contrast

Controls the color contrast. 100% is normal, less washes out the colors, more makes the differences in color sharper.

#### Saturation

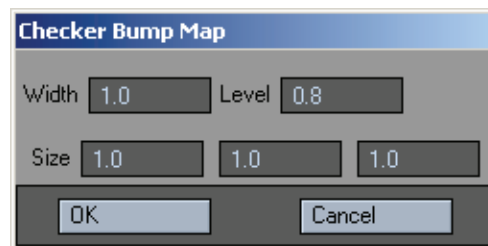
Controls the color saturation. 100% is normal, less changes the colors towards black & white, more makes the colors appear brighter.



**Note:** Setting CCTV such that it views the polygon it is shading does not show the polygon. This is a known limitation in LightWave3D where shaders can't "see" themselves when raytracing.

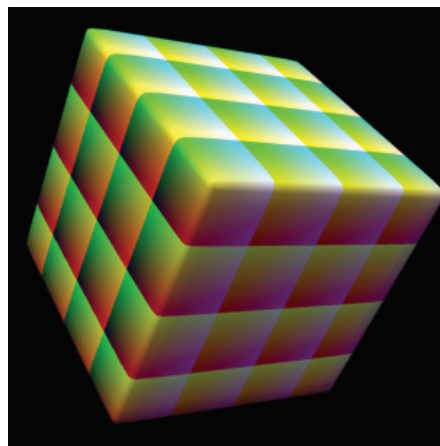
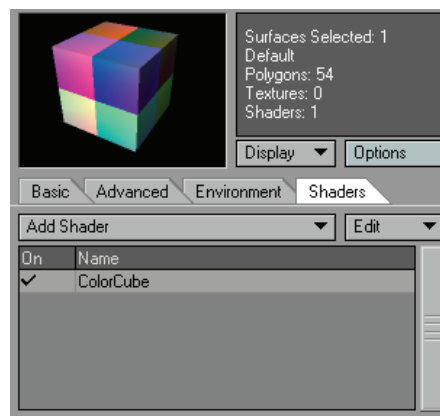
## Checker

This shader puts a **Checker Bump Map** on the surface and offers options for **Width**, **Level**, and **Size** of map.



## ColorCube

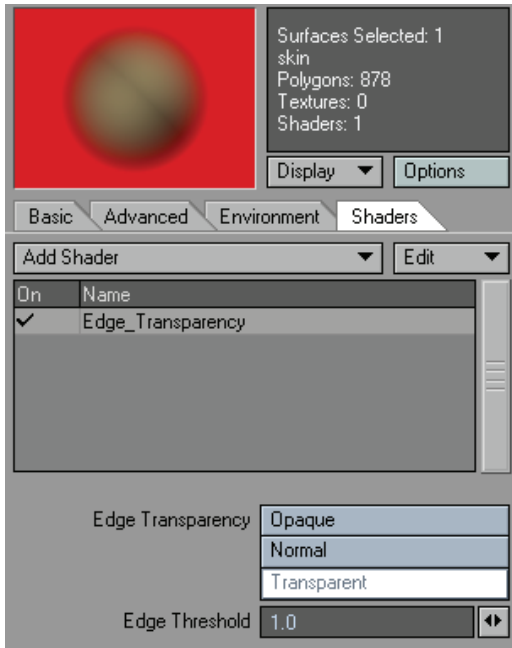
It translates the 3D coordinates or the surface spot being rendered into that spot's **RGB Color** value. The colors repeat at one-meter intervals.





## Edge\_Transparency

The **Edge Transparency** shader affects the degree of clarity and definition for the polygon edges belonging to a transparent object. **Opaque** creates an adjustable edge. **Normal** creates a solid edge. **Transparent** causes the edges of the object to blend into its surroundings.



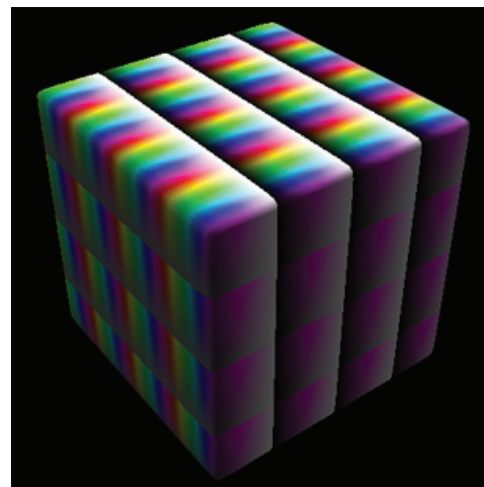
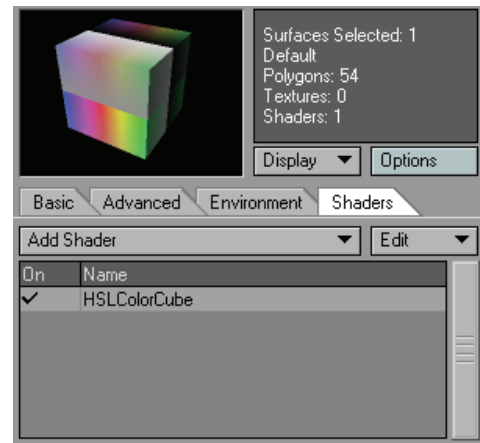
**Edge Threshold** is available for **Opaque** or **Transparent Transparency** settings only. It determines the amount of blending between the surface color and the transparent edge of the object surface. This transition zone may be wide and softly blended over the face of the surface, or it may be sharp and seen as a thinner line. **Edge\_Threshold** is normally set at 1.0. Use lower values for a sharper transition. Use higher values for a softer transition.



NOTE: When you fine-tune for glass-like objects, we recommend you use **Limited Region (Camera Properties)** to render a small portion of the image at full resolution. Test rendering at lower resolutions may cause jaggy edges that won't be visible at your higher final resolution.

## HSLColorCube

It translates the 3D coordinates or the surface spot being rendered into that spot's **HSL (Hue, Saturation, Lightness) Color** value. Like **Color Cube**, the colors repeat at one-meter intervals.







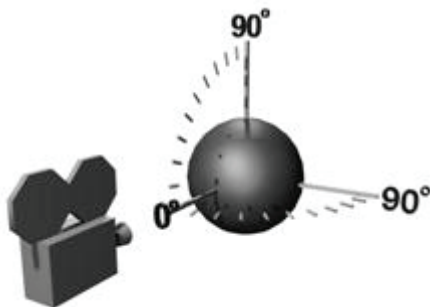
## Fast Fresnel

In the real world, the angle between the viewer and a surface affects the amount of light that is reflected and refracted. For example, a lake is often nearly clear close to the viewer and gets gradually more reflective as the viewer moves farther away.

The **Fast Fresnel** shader works in combination with the **Basic Surface** parameters that you set and then modifies those settings based on viewing angles and the **Fast Fresnel** parameters.



The **glancing angle** is measured from the surface normal, thus 0 degrees is any surface normal that points directly at the camera and 90 degrees refers to any normal that is perpendicular to the camera.

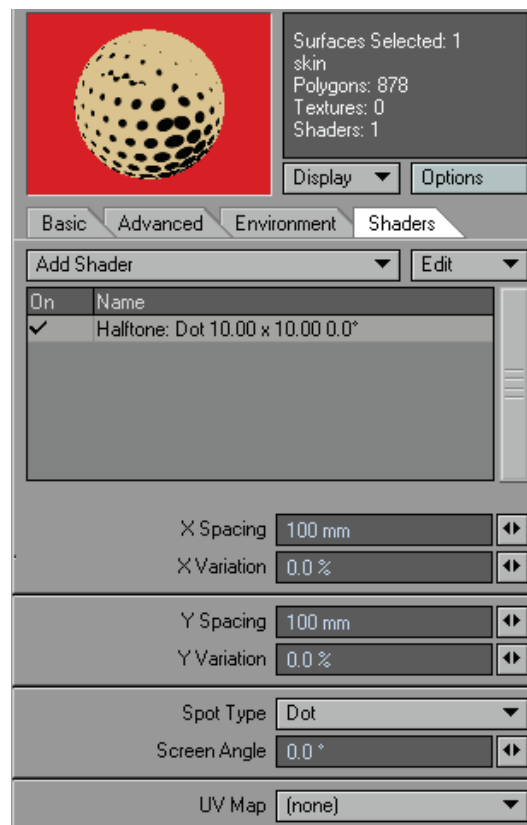


The regular surface value always occurs looking directly at the surface. However, you can adjust the point when the value begins to migrate towards the **Fast Fresnel** settings using the **Minimum Glancing Angle**. At a 90-degree angle, the attribute becomes equal to the value set on the **Fast Fresnel Panel**. Essentially, the regular **Surface** settings change to the **Fast Fresnel** settings as the angle of incidence goes from the **Minimum Glancing Angle** to 90 degrees. (Although, the **Minimum Glancing Angle** can be 0 to 89 degrees, you'll usually want this at or near 0 to create a wide range for the effect.)

Higher glancing angles cause parameters that default to 100, like **Reflectivity**, to gain intensity while they cause parameters that default to 0, like **Transparency**, to naturally lose intensity.

## Halftone

In print, halftone screens are made up of dots that control how much ink is deposited at a specific location. Varying their size and proximities creates the illusion of variations of continuous color.



The **Spacing** and **Variation** settings set the spacing of the pattern and a percentage that you can use to vary it (for randomness), respectively. You can choose different pattern shapes with the **Spot Type** pop-up menu. The **Screen Angle** value lets you change the angle of the pattern from the true horizontal.

If you specify a **UV Map**, the shader determines a direction in the surface to align the lines or the crosshatching. **Halftone Shader** is calculated like a **Cubic** texture, if no **UV Map** is specified.

## LW\_HyperVoxels\_Shader

This surface shader allows object surfaces to receive shadows from **HyperVoxels**, as well as reflect and refract them. If you elect to have the surface receive **HyperVoxel** shadows, you have the option of using the object's **Shadow** settings (i.e., **Self**-, **Cast**-, **Receive Shadow (Object Panel)**).

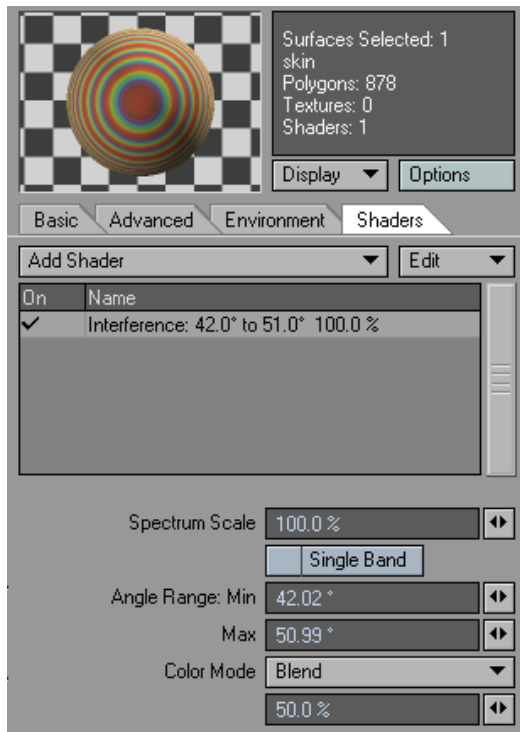
**Shadow Strength** makes shadows more or less opaque. The **Fast Shadows Render Mode** does not account for hypertextures in the shadows. **Accurate** renders the shadows including the effect of hypertextures. To use the **Reflect** and **Refract** options, the surface must be reflective or transparent; however, the **Reflection Option (Surface Panel)** setting is not considered.





## Interference

The **Interference** shader adds the sort of distortion seen on an oil slick. This **interference** pattern is caused by the light reflecting between two layers of various thickness. In the case of an oil slick, light reflects between the water and the oil. This often appears rainbow-like, where you can see all spectral colors swirling about. This shader adds all spectral colors in a banding fashion and uses incidence angles to vary which colors are visible.



**Spectrum Scale** determines how far through the color spectrum the shader will travel across the slope of the surface. This is dependent on the **Min** and **Max Angle Range** settings. For example, the default settings of **Spectrum Scale**=100%, **Min**=42, and **Max**=50 tell the shader to travel through the entire spectrum (100%) as the angle of incidence changes from 42 degrees to 50 degrees, or a delta of 8 degrees. The spectral range colors are red, orange, yellow, green, blue, indigo, and violet. If you change only the **Spectrum Scale** to 50%, the surface travels only through red, orange, yellow, and green, across the same angle.

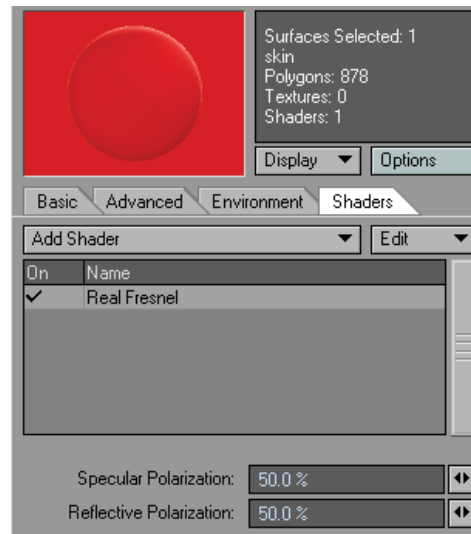
Activating the **Single Band** option restricts the spectral change to a single ring between the **Min** and **Max** angles. In effect, this keeps the texture from repeating across the entire surface.

The **Add Color Mode** adds the **Color** values of the interference pattern to the original surface colors. If a pixel was originally shaded at 100, 100, 100, and the interference color for that pixel was to be 0, 0, 100, the resulting pixel value will be 100, 100, 200. The **Multiply** option multiplies the original pixel values by a number between 0 and 1. If the original pixel value was 255, it is multiplied by one. If it was 0, the new pixel value is multiplied by zero. Intermediate values are altered on a sliding scale.

The **Blend Color Mode** blends the pattern with the original surface attributes using the percentage field. At 50%, the default, the interference pattern is seen on top of the original surface with a strength 50% of its own **Color** values.

## RealFresnel

The **RealFresnel** shader is similar to **Fast Fresnel**, but based on real physics and thus features few user-definable controls. It is essentially set up to create a transparent item by calculating **Falloff** for the **Transparency** value with the **Fresnel** equation.



**Specular Polarization** and **Reflective Polarization** are the values for those surface attributes that will be used when the camera is perpendicular to the surface.

**RealFresnel** is more accurate than **FastFresnel**, but will probably require more time and effort to find settings that will work for your scene.

## LScript

This shader allows you to add an LScript to a surface.

## LScript/RT

This shader allows you to add a compiled LScript to a surface.

## LW\_Rust

**LW\_Rust** is a legacy shader that evaluates the accessibility of polygons to be shaded.

This is accomplished by determining when a defined polygon axis intersects (i.e., hits) another polygon within a defined radius. The direction is set by the mode you use. You can use this plugin to add rust as well as dust to objects.



NOTE: This function can be created with the use of **Gradients** in the **Texture Editor**. This shader is here to give you the ability to render old LightWave scenes.



## LW\_Snow

**LW\_Snow** is a legacy surface shader which controls the surface and shading based on the slope of the geometry. This provides the ability to have a white color on areas creating a snow effect.



NOTE: This function can be created with the use of **Gradients** in the **Texture Editor**. This shader is here to give you the ability to render old LightWave scenes.

## LW\_Water

**LW\_Water** is a legacy shader that lets you quickly add realistic water surfacing.



NOTE: This function can be created with the use of **Gradients** in the **Texture Editor**. This shader is here to give you the ability to render old LightWave scenes.

## Stress Map

Stress Map makes more realistic, dynamic wrinkles. It does this by altering the degree of bumpmapping used based on the amount of local polygon distortion. So for example, when an arm is bent, wrinkles applied as a bumpmap become more pronounced around the joint, where the mesh is being squeezed. Or, if the mesh is stretched the wrinkles will disappear.

Stress Map can also be used to alter the color of a surface based on how much a polygon has expanded or shrunk due to some distortion. This could be used to simulate the whitening of the skin as it is stretched, or to indicate areas where the mesh is being stressed.

Stress Map works by modifying the degree of bumpmapping or surface color based on how much the mesh is stretched or shrunk around the point being shaded. The distortions may be due to scaling in Layout, the action of Bones, or the application of a morph map for example.



Stress Map works in combination with the the surface's bump mapping setting. Think of the bump map as indicating the potential for wrinkles to form. Stress Map then realises that potential to varying degrees based on how much polygons of the mesh around the point being shaded have grown or shrunk. The distortion is computed by comparing the undistorted mesh with the mesh after the application of scaling, bones, morph maps, etc.

### Skin Area

The skin area is the assumed area of the skin when wrinkles are included. This is generally set no smaller than the area of the undistorted mesh. The skin area is a measure of local skin surface area. It may vary over the mesh.

The value set for Skin Area should be considered in combination with the Skin Area Scale setting.

#### Base polygon area

Sets the Skin Area equal to the area of the polygons of the undistorted mesh.

#### Custom...

Some constant value, envelope, or texture can be used.

### Skin Area Scale

The value obtained from the Skin Area setting is multiplied by the Skin Area Scale to get the skin area that is used in the computations.

By setting the Skin Area to Base polygon area, Skin Area Scale can be used to easily make the skin slightly larger than the polygon mesh area. Skin Area Scale can also be used to quickly tweak the skin area to give a satisfactory amount of apparent skin.



## Stress Map Scale

Scales how deep wrinkles appear to be. This is usually kept at 1.0, but other values can be used to make wrinkles more or less pronounced.

## Wrinkle Factor

When bumpmapping, the surface normal is perturbed by a small bump vector to give the appearance of bumps. The Wrinkle Factor scales this bump vector. It is this setting that causes wrinkles and bumps to become more or less pronounced as a function of surface distortion.

When set to 0, you won't see any bump mapping. When set to 1, the result is the same as the original bump mapping. Wrinkle Factor is typically set to be a function of Stress Map, either directly or through a texture gradient.

## Stress Map

Stress Map is a value that represents by how much bumps should be scaled. Mathematically it is defined as:

$$\text{Stress Map} = \left( \frac{\text{Skin Area} * \text{Skin Area Scale}}{\text{Distorted polygon area}} - 1 \right) * \text{Stress Map Scale}$$

If the scaled skin area is equal to the distorted polygon area, Stress Map is equal to zero (the skin is perfectly taut). It increases as the skin area increases or the polygons are made smaller.

## Clipped Stress Map

The Stress Map value can become less than zero when the distorted polygon area is larger than the scaled skin area. This is somewhat unphysical as it implies that the skin is stretched beyond breaking point.

Clipped Stress Map is equal to Stress Map, but limited to a value no smaller than zero.

## No Wrinkles

Sets the wrinkle factor equal to zero. Handy if you want to use Stress Map only to change the skin color.

## Skin Color

The color of the skin. It is mixed with the surface color according to the Skin Opacity setting.

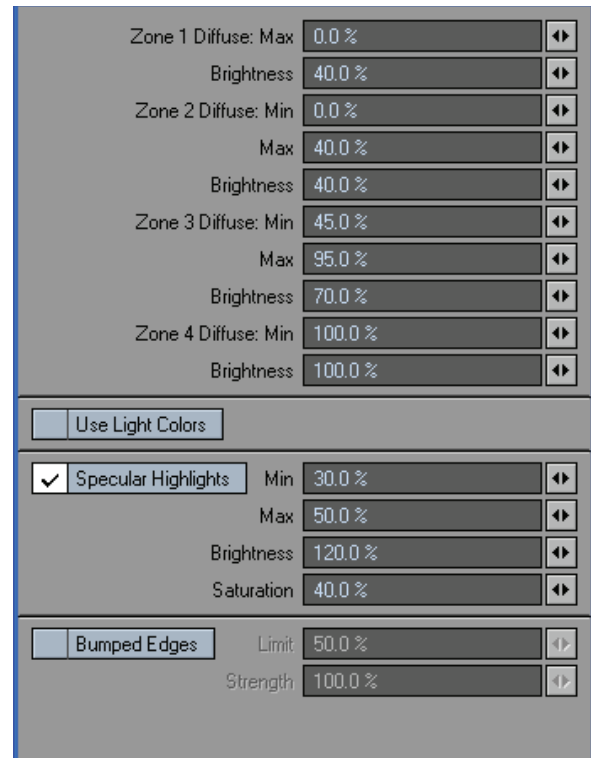
## Skin Opacity

Skin Opacity determines how much of the color obtained from the Skin Color setting to use in the mix with the surface color.

When set to 0, none of the skin color is used. When set to 100%, the skin color is used exclusively. 50% would result in an even mix of surface and skin color.

## Super Cel Shader

**Super Cel Shader** alters the shading algorithm to make large bands of solid color across a surface — like a cartoon — rather than a smooth gradient between light and dark areas. This shader should be used with **Silhouette Edges**, **Unshared Edges**, **Sharp Creases**, and **Surface Borders** (Object Properties Panel, Edges Tab).



## Defining Color Zones

**Super Cel Shader** has four possible color zones (1 through 4). An area's amount of **diffuseness** (i.e., brightness) determines how many zones the surface is broken down into. The zones go from darker to lighter, from 1 to 4, respectively. Essentially the **Min** and **Max** settings define the color zones.

**Min** and **Max** values should range from 0 to 100% and must increase progressively. For example, the **Zone 3 Diffuse: Min** must be higher than or equal to the **Zone 2 Diffuse: Max**. If the **Min** value is the same as the lower zone's **Max**, a hard edge is created between the zones. Separating the values will smooth the color transition.

The default settings for the panel disable Zone 1 by making the **Zone 1 Diffuse: Max** value equal to zero. Note that the **Zone 1's diffuse minimum** is always zero and **Zone 4's diffuse maximum** is always 100% — there are no input fields for these.

Notice the 5% difference between the **Zone 2 Diffuse: Max** and the **Zone 3 Diffuse: Min** values, creating a slight smoothing between the color transition.



## Brightness

The **Brightness** settings control how bright the corresponding zone will be. These settings are totally independent; however, in most cases, you want them to increase progressively. Values should range from 0 (black) to 100% (brightest), although higher settings are possible.

## Colored Lights

Activating the **Use Light Colors** option will tint the surface with the light color. Normally, the normal Surface Color attribute determines the surface's color.

## Specular Highlights

If **Specular Highlights** is not active, the **Specular Highlights** of the surface appear normal (i.e., as without **SuperCelShader**). Activating this option results in **cartoony-looking Specular Highlights**.

The **Min** and **Max Specular** settings define which part of the **Specular Highlight** to show. A value of 0% equates to its darkest part and 100% equates to its brightest part — this range can often be hard to see in a normal **Specular Highlight** because they are naturally bright. Making the **Min** and **Max** values the same creates a hard edge to the highlight, while separating the values creates a smoother transition. The higher the **Min** value, the smaller the **Specular Highlight** will be.

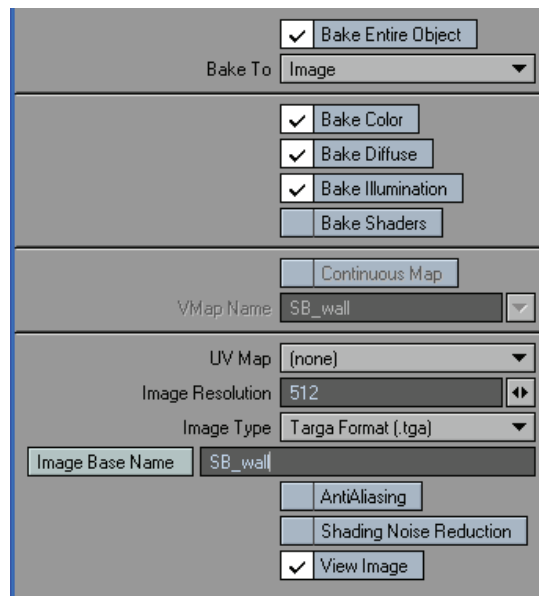
**Brightness** controls how bright you want the overall highlight to be. **Saturation** determines how much of the surface color you want to saturate the highlight.

In some cases, you want to **soften** the edges of cel-shaded surfaces to avoid the harshness that can sometimes occur. Activating the **Bumped Edges** option makes the edges **soft and fuzzy**. This has no direct effect on the shading of the zones, other than near the edges where the surface normal is angled toward 90 degrees from the camera — like the edges of the ball in the preview window.

**Limit** basically defines the minimum angle — of the surface normal relative to the camera — to be affected. 100% is a surface normal pointing at the camera and 0% is perpendicular to the camera. As **Limit** is set towards 100%, more edges are affected. **Strength** determines how much to affect the edges. Values greater than 100% begin to **eat away** the edges.

## Surface Baker

The **Surface Baker** surface shader allows the **baking** of lights, shadows, textures, and so on. You can save this information to an image to **UV Map** back onto the object, or you can apply the information directly to the object's vertices. After you use **Baker**, you can see complex textures in real-time in your viewports!



If the surface has sharp creases or is not smooth, you may not want to interpolate across polygon boundaries. Uncheck **Continuous Map**: this essentially turns off polygon **Smoothing** and the map will change sharply at each vertex. You might, for example, keep this option off if you were **baking** a room and did not want to smooth between the walls.

**Bake Entire Object** is a time-saving option that lets you add **Baker** to a single surface and create its results for all of the object's surfaces.

You can independently choose to bake in color, diffuse shading (e.g., bumps and diffuse textures), illumination (all the lighting from the scene, including shadows, projection images, **even radiosity and caustics**), as well as other shaders in the surface baking computation.

Baking the illumination takes the lighting environment into consideration for the **diffuse reflection**. If **Bake Illumination** is **off**, the surface is calculated in an environment without lights and 100% white ambient color. If it is the only option **on**, then you bake only the light intensities that reach the surface and discard the **Surface** settings.



NOTE: When baking surface shaders, **Baker** can be anywhere in the shader list. It does not need to be at the top or bottom.

## Baking Tips

Due to the complexity of the computations, only polygon faces with three or four vertices can be baked. Other faces are ignored (tripling your polygons will ensure you meet this requirement). Moreover, some portion of the surface you are baking must be visible to the camera at the time of rendering. Only some portion of the surface needs to be visible. (If the surface is not visible, the plugin does not get executed during the render.)

You cannot bake elements that depend on the relative position of the camera and the lights, like incidence angle gradients, transparency, refractions, and reflections. These will not bake correctly. However, you may use the incidence angle from a light as the **Input Parameter** for gradients (but not the incidence angle from the camera).



NOTE: Baking to a **Vertex Color Map** saves RGBA data, using floating point values for RGB, not limited to 1. The **Alpha** value is 1.0. Baking to an image can also save RGBA data, using floating point values for RGB, not limited to 1. However, the **Alpha Channel** is saved only if the file format supports it and the **Wireframe** is saved in the **Alpha Channel**.

## Image Mode

With the **Image Mode** (**Bake To** pop-up menu), each polygon is **painted** onto the final image based on its **UV** coordinates. Once **Baker** creates the full image, you simply **UV Map** it onto the surface, as you would normally. With the proper **UV Map**, the surface is seamless.

The key to pulling off this effect is starting with a good **UV Map**. If the map cannot acceptably line up the UVs with the appropriate parts of the image, you will get unacceptable results.

The **Image Resolution (Image Mode)** sets the pixel width and height of the image to be created.

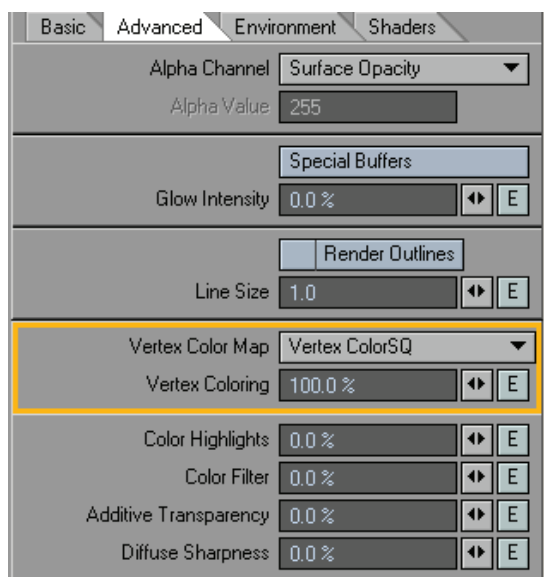
Use the **Image Base Name** button to navigate to a directory to save your image files. Enter a base filename. A frame counter and format extension will be added automatically. Because the frame counter will be incremented, you can save an image sequence of your surface.

If you choose an **Image Type** that supports alpha channel, the **Wireframe** view of the mesh is saved to the alpha channel.

There are options at the bottom of the panel to toggle **Antialiasing** and **Shading Noise Reduction** algorithms on or off. If **View Image** is checked, baked images will be displayed after rendering.

## Object Mode

With the **Object Mode** (**Bake To** pop-up menu), the surface is sampled at each vertex and assigned to a vertex-shading **V Map** called a **Vertex Color Map**. Essentially, color information is stored with the points and thus becomes part of the object. To use this map on the surface, go to the **Advanced Tab** and select it from the **Vertex Color Map** pop-up menu. Set the blending percentage in the **Vertex Coloring** field.



The surface area between each vertex is interpolated, which obviously means the result can be only an approximation of the original surface. A higher density of points (i.e., vertices) in the surface will increase accuracy.



NOTE: Since the **Object Mode** samples only at each vertex, it computes much faster than the **Image Mode**.

### To use Baker:

**Step 1:** If you plan to use the **Image Mode**, you must first create a **UV Map** in Modeler.

**Step 2:** In Layout, add the **Baker** surface shader to the target surface.

**Step 3:** Select the surface attributes you want to **bake in**.

**Step 4:** Select the **Bake To** mode.

**Step 5:** Set the **Vertex Color Map** name (**VMap Name**) or **Image File** name, depending on mode used.

**Step 6:** Close the panel.

**Step 7:** Render a frame (**F9**).

**Step 8:** Remove the **Baker** shader from the surface or deactivate it.

## Instant Radiosity

You can use **Baker** and radiosity to compute an **Accessibility Map**. An **Accessibility Map** shows how accessible a point is on the surface, which lets you create **dirt**, **weathering**, or radiosity lighting effects. Surfaces are normally darker in grooves and creases, not only because less light reaches these areas, but also because dirt has a tendency to accumulate there.





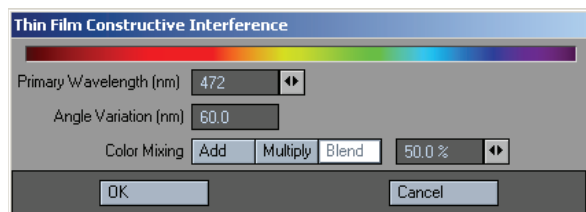
## Surf Mixer

**Surf Mixer** is a shader that allows you to take the surface that you have applied this shader to, and mix another surface in the scene with that surface.



## Thin Film

Similar in effect to **Interference**, the **Thin Film** shader also changes the color spectrum based on the surface's angle to the camera. It can be used for effects like an oil film on water. **Primary Wavelength** is the color in the spectrum that the shader will use as its base color. You may either enter the wavelength value or simply click on a color in the spectrum. **Angle Variation** is the angle at which the colors will start to shift.

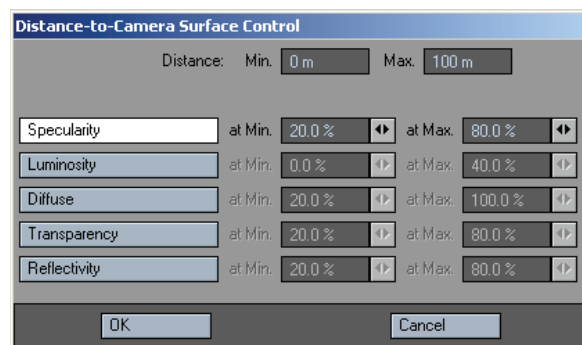


When **Color Mixing** is set to **Add**, it adds the color values of the interference pattern to the original surface colors. If a pixel was originally shaded at 100, 100, 100, and the interference color for that pixel was to be 0, 0, 100, the resulting pixel value will be 100, 100, 200. The **Multiply** option multiplies the original pixel values by a number between 0 and 1. If the original pixel value was 255, it is multiplied by one. If it was 0, the new pixel value is multiplied by zero. Intermediate values are altered on a sliding scale.

When **Color Mixing** is set to **Blend**, it blends the pattern with the original surface attributes using the percentage field. At 50%, the default, the interference pattern is seen on top of the original surface with a strength of 50% of its own color values.

## Z Shader

**Z Shader** lets you vary the values of certain surface attributes over a specified distance from the camera.



Define the distance from the camera range using the **Min** and **Max** fields. Activate the attribute you want to vary by selecting its button. Enter the values for the minimum and maximum distances in the fields provided. In-between values will be interpolated.

### Normal Color

This shader varies the surface color according to the angle of the polygon's surface normal.

### Pulse

This shader is an Iscript application that cycles the luminosity and diffusion (inversely) of a surface so that the surface appears to "pulse".

### Steamy Shader

This surface shader will allow the surface to which it is applied to reflect and refract selected **Steamer** items. You can also make the surface receive shadows from **Steamer** objects. The **Accurate Render Mode** will yield better looking results at the expense of rendering time.

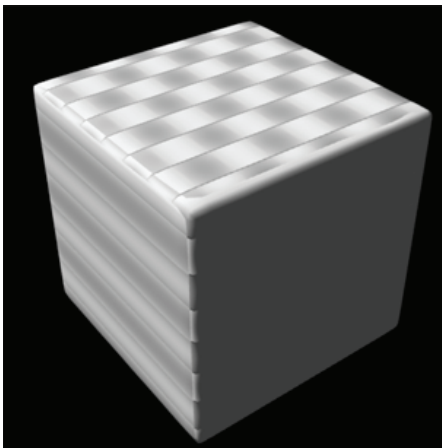
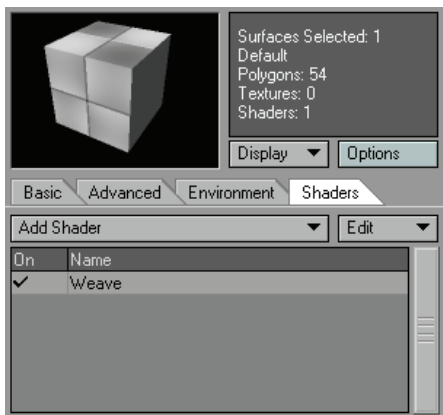


NOTE: **Hypervoxels** replaced **Steamer** in newer versions of LightWave. This shader is needed for older scene files that have **Steamer** applied to them.



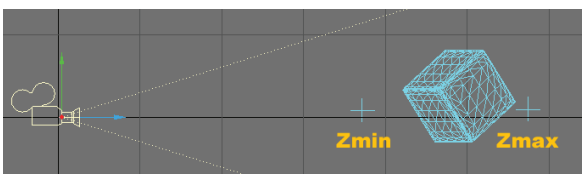
## Weave

This shader puts a **Bump Map** on the surface. The **Options Panel** has input for the **Width**, **Level**, and **Size** of the map.



## Zor

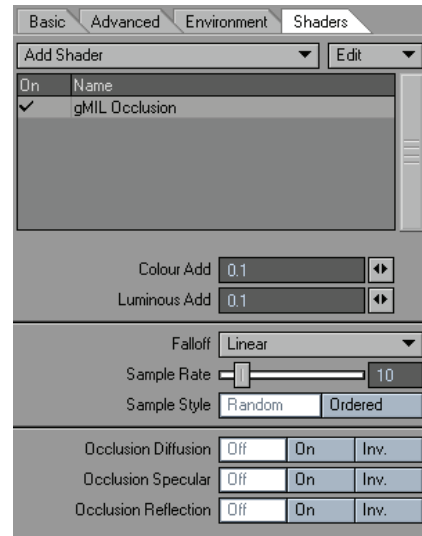
This shader varies the opacity of your surface on the **Z-axis**. To use **Zor**, add two Null objects and name the **Zmin** and **Zmax**. Activate the **Zor** plugin for the surface of some object in the scene. Place **Zmin** at the edge of the surface nearest the camera and **Zmax** at the furthest edge. When the frame is rendered, the edge nearest the camera is completely transparent; the furthest edge is completely opaque.



You can place **Zmin** and **Zmax** anywhere in the scene and even animate them; however, **Zmin** should always be closer to the camera than **Zmax**. **Zor** surfaces closer to the camera than **Zmin** will be transparent, everything further than **Zmax** will be opaque.

## gMIL

This is a surface shader for LightWave that simulates global illumination, based on the backdrop on a surface by surface basis. It adds color or illumination to the surface's normal shading received by scene lights. The shadow it receives from other objects is calculated by sampling an imaginary lighting sphere around your scene, either random or ordered. Objects that cast shadow on the (**gMil**) surface can optionally occlude **Diffuse** and/or **Specular** lighting and reflection, this either in a additive (**ON**) or negative (**INV**) fashion.



By using **gMil** in combination with **BRDF** you can put up a relation between your scene lighting (also put some ambient lighting) and global illumination to change the shadow appearance.

### gMil Settings:

**Color Add** — Adds color from the backdrop.

**Luminous Add** — Adds some luminosity (brightness).

**Falloff** — Doubles the shading for the diffusion/darkens the shadows.

**Sample Rate** — Quality level (impacts render times).

**Sample Style** — Changes the type of shadow from dithered noise to a smooth smudge-like noise.

**Occlusion** — Lets you turn on/off each attribute affected by **gMil**.







---

## Chapter 25: Node Editor

---



## Node Editor

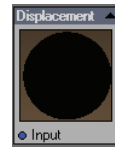
This chapter is divided into two sections. First an overview of the Graphical User Interface (GUI) and some of the conventions used when working with nodes in the Node Editor. In the second section we will discuss each node individually and in detail in reference guide format for easy look-up of any node's function, purpose and the various inputs and outputs associated with each.

The node editor is an alternative and superior way of shading and texturing the geometry created in the modeling process. Node based editing systems such as commonly found in many high end 3D computer graphic applications, have established themselves as an industry standard tool for shading and texturing. The Node Editor in LightWave 3D is superior to many others combining the ease of use associated with LightWave 3D with the power and flexibility required by high end industry professionals. As we will see, it is easy enough for a beginner to begin shading and texturing objects within just a few minutes and powerful enough for advanced users to design their own custom shaders and textures either by constructing special purpose node networks in the user workspace area or by using the LightWave 3D SDK that is offered free from NewTek Inc.

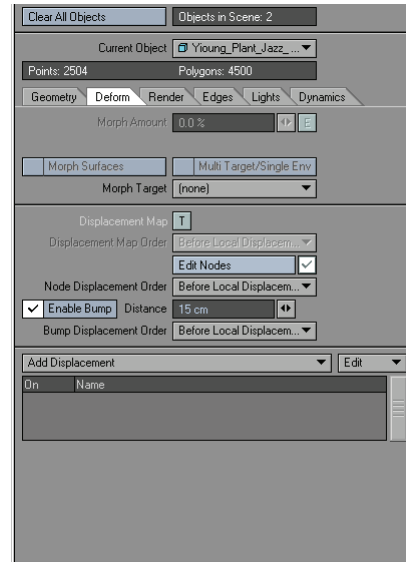
The Node Editor currently exists in three different places throughout the LightWave 3D working environment. Each environmental instance of the Node Editor is identical in operation and technique except for one key aspect - the Destination node, sometimes also referred to as the Root node.

There is always only one destination node for any attribute (volumetric lighting, vertex deformation, or shading and texturing) that nodes are applied to. Here are the three locations and an example of what each of the Destination nodes look like.

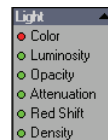
## Displacement Destination Node



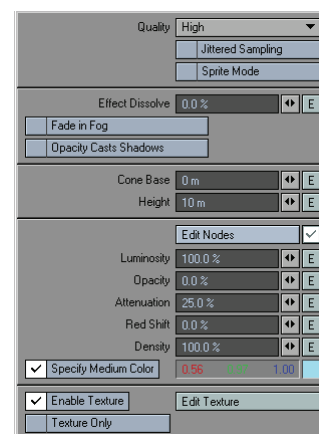
This will be the Destination node if the Node Editor is opened via the Edit Nodes button found in the Object Properties panel under the Deform tab (Accessed by pressing [O], [p] ).



## Light Destination Node



This Destination node will be present if the Node Editor is opened by clicking the Edit Nodes button located in the Volumetric Options panel contained in the Light Properties panel's Basic tab (Accessed by pressing [L], [p] ).

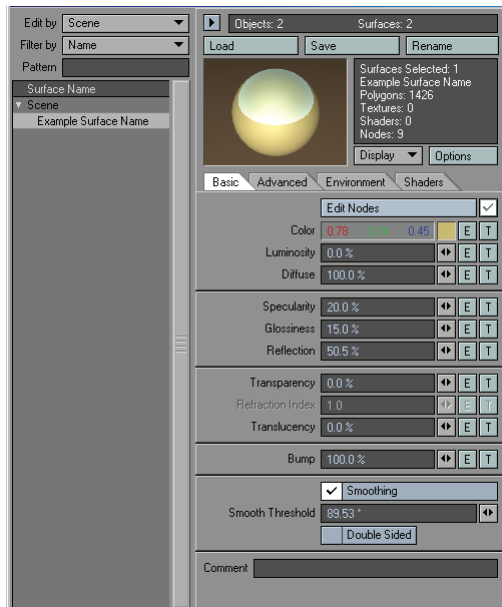




## Surface Destination Node



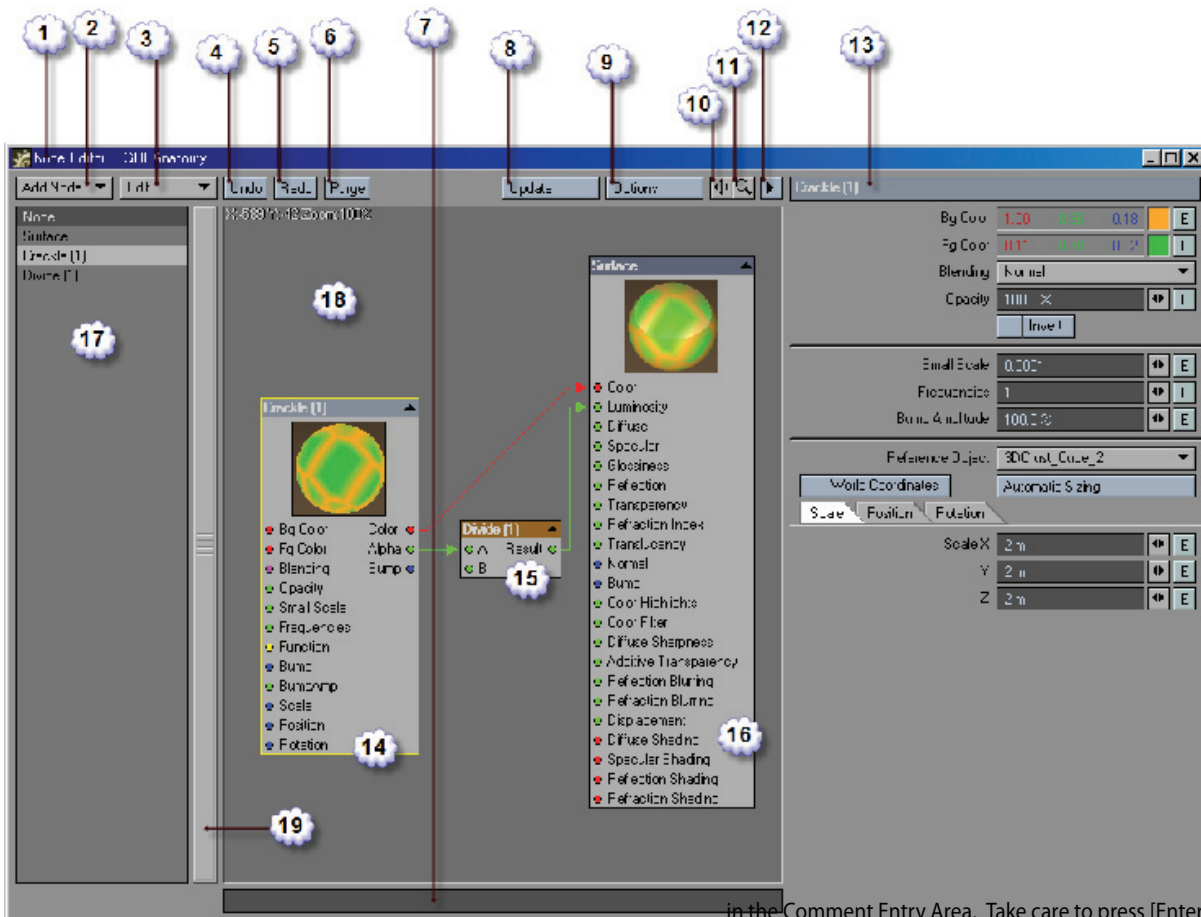
This is the Destination node you will be using if you open the Node Editor by clicking on the Edit Nodes button found in the Surface Editor panel [F5].



For the purpose of explanation we will be focusing almost exclusively on the Node Editor as it appears when opened from the Surface Editor panel. Both the Surface Destination node and the other Destination nodes will be covered in detail in the “Destinations” section later in this chapter.

## GUI Anatomy

Let's take a look at the graphical user interface (GUI) for the new Node Editor. Use the numerical look-up call-outs to identify each element and read its description in the corresponding text below.



# 1) Windows Drag Bar.

Here you'll see "Node Editor" to indicate the panel type followed by the current surface name or the property if you are working with displacements from the Object Properties panel or Volumetric Lights from the Volumetrics Options panel.

## 2) Add Nodes Pull Down Menu.

Use this menu to add nodes to your Workspace (See #18 for a description of the Workspace).

## 3) Edit Pull Down Menu.

Contained in this menu are all of the edit functions for working in the Node Editor GUI. From copying and pasting to importing and exporting the GUI manipulation tools you'll be using most will be found in here.

This menu is also available by right clicking on top of any node in the Workspace Area or listed in the Scrollable Node List.

## 4) Undo Button

Click on this button or press [Ctrl+z], to undo the last Node Editor operation you performed.

## 5) Redo Button

Click on this button or press [z], to redo the last undo operation you performed.

## 6) Purge Button

Click on the purge button to discard all of the undo history currently in memory for the node editor.

## 7) Node Comment Entry Area

Add comments to any node by selecting the node and typing

in the Comment Entry Area. Take care to press [Enter] when finished or your comments will not be saved with the node.

## 8) Update Button

Click on the update button to force a manual refresh of any preview spheres displayed in the Node Editor panel.

## 9) Options Button

Clicking on the options button will display the option settings panel for the Node Editor.

## 10) Workspace Area Pan Gadget

Using the Pan Gadget (or widget) or by alt-dragging in the Workspace area, will allow you to scroll the Workspace area.

## 11) Workspace Area Zoom Gadget

Use this gadget to zoom in or out of the workspace area. This is useful when working with or constructing large networks and will allow you to quickly locate nodes and network segments that may be scrolled out of view.

## 12) Embedded Edit Panel Extension Gadget

Use this gadget to pop open the embedded edit panel which can be used in place of floating edit panels when editing node properties. To edit node properties with the extension area closed simply double click on the node you want to edit.

## 13) Embedded Edit Panel Extension Area

With this area extended any selected node's properties that has an edit panel can be edited in this embedded panel area.

## 14) Selected Node

This is an example of a selected node.

## 15) Unselected Node

This is an example of an unselected node.

## 16) Surface Destination Node

This is the Surface Destination node. Making connections to this node from other nodes in the Workspace area define a surfaces' properties.

## 17) Scrollable Node List

In this area you will find a list of all the nodes currently added to your Workspace area. You can use this list box to select, multi select, or edit, any nodes you have added.



### 18) Workspace Area

Use the workspace area to construct networks by adding nodes and connecting them together.

### 19) Node List Scrollbar

When the length of the list of added nodes is longer than your current Node Editor panel is high, use this scrollbar to scroll list of nodes.



NOTE: 14, 15, and 16 together form what is called a Network. You can think of node networks as being very much like computer networks - with connections, input, output and the most important bit: results!

## Menus and Hot Keys

The menus and hot keys described here apply only when the Node Editor window is active. To activate the Node Editor window if it is not active simply click anywhere on its interface. To display the Node Editor window if it is not displayed simply click on the Edit Nodes button on the options panel you're working with. Remember, the Node Editor lives in several places throughout the LightWave3D application.

Edit	ret
Rename	r
Preview	▶
Copy	^c
Cut	^x
Paste	^v
Delete	del
Select All	a
Clear Selection	d
Invert Selection	i
Fit All	+F
Fit Selected	f
Disconnect inputs	+D
Expand	+E
Collapse	+C
Export Selected	
Import Nodes	

### Edit [ret]

Pressing "Enter" or selecting Edit from the Edit menu, with any node selected and the embedded edit panel closed will open that nodes floating Edit Panel. In ancient times the Enter key was also referred to as the Return key.

### Rename [r]

Pressing the "r" key or selecting Rename from the Edit Menu with any one node selected will open a Rename Node requester that will allow you to rename the node. The name of a node appears in its title bar area.





## Preview (submenu)

Preview options are context sensitive. Depending upon what types of output the selected node offers, any of several preview options will be available. The preview option that is selected will determine what the preview sphere displays if the Disable Previews checkbox is unchecked in the Node Editor's Options panel.

The various possible types are:

Color	Color Channel
Alpha	Alpha Channel
Luma	Luminance Channel
Bump	Bump Direction in three colors
Result	Line Graph Display
Normal	Normal Direction in three colors

## Copy [Ctrl+c]

Pressing control and "c" together or selecting Copy from the Edit Menu will copy any selected nodes into the copy buffer. If text is selected in the Note Comment Entry Area, numerical input areas or etc then the selected alphanumeric string will be copied into the copy buffer.

## Cut [Ctrl+x]

Pressing control and "x" together or selecting Cut from the Edit Menu will cut any selected nodes into the copy buffer. If alphanumeric string data is selected in the Node Comment Entry Area, numerical input areas, or etc then the selected alphanumeric string will be cut into the copy buffer.

## Paste [Ctrl+v]

Pressing the control and "v" keys together or selecting Paste from the Edit menu, will paste any selected nodes into the Workspace area that are currently contained in the copy buffer. A separate copy buffer is simultaneously available for both nodes and for text so that you could for example, copy some text, copy a node, and then paste the text or the node in their attributed areas without losing the information in either of the two copy buffers.

## Delete [Del]

Pressing the delete key or selecting Delete from the Edit menu, will delete any nodes or alphanumeric data that you may have selected. This delete operation only removes the item from the graphical user interface. No menu entries or disk based items will be affected. Delete operations do not place the deleted items into the copy buffer.

## Select All [a]

Pressing the "a" key or choosing Select All from the Edit menu, will select all the nodes currently in the Workspace area.

## Clear Selection [d]

With the Node Editor window active, pressing the "d" key or selecting Clear Selection from the Edit menu, will deselect any selected nodes in the Workspace area.

## Invert Selection [i]

Pressing the "i" key or choosing Invert Selection from the Edit menu, will invert the current selections. If no nodes are selected when "i" is pressed then all nodes will become selected.

## Fit All [F]

Pressing the "Shift" and "f" keys together or selecting Fit All from the Edit menu, will fit and center all of the nodes currently in the Workspace area. If all of the nodes cannot be displayed in the Workspace area at 100 percent zoom then the workspace area will be automatically zoomed out in order to fit all the nodes currently in the Workspace area.

## Fit Selected [f]

With the Node Editor window active pressing the "f" key or selecting Fit Selected from the Edit menu, will fit and center any node or nodes currently selected in the Workspace area. If all selected nodes cannot be displayed in the Workspace area at 100 percent zoom ratio then the Workspace area will automatically be zoomed out in order to fit all of the nodes currently selected into the Workspace area.

## Disconnect Inputs [D]

Selecting one or more nodes in the Workspace area and pressing the "D" key or selecting Disconnect Inputs from the Edit menu, will disconnect any inputs that are connected to the selected nodes. Output connections from a selected node will not be disconnected. You can think of this as disconnecting all connections made to the left hand side of the node.

## Expand [E]

Pressing the "E" key on the keyboard or selecting Expand from the Edit menu, will expand any selected nodes that are currently collapsed.

## Collapse [C]

By pressing the "C" key on the keyboard or selecting Collapse



from the Edit menu, will collapse any selected nodes that are currently expanded.

## Export Selected

Choosing Export Selected from the Edit menu will open a Save file requester that will allow you to save the node or group of nodes you have selected as a .nodes file. A ".nodes" file is a structured text file containing the names of all the nodes you had selected when you saved the file, any comments that you had entered for each of the nodes selected, and any of the connections that existed between the selected nodes. It basically exports a complete snapshot of the nodes you had selected when you exported.

It is important to note that the Destination node will not be exported nor any of the connections that existed to the Destination node when the file was saved.

This is a very useful feature allowing you to create and maintain directory based libraries of various network segments or entire networks. ".node" files can be loaded and saved interchangeably between any of the three Node Editor windows for Displacements, Volumetric Lights, and Surfaces.

## Import Nodes

Selecting Import Nodes from the edit menu will open a Load File requester allowing you to load any saved ".nodes" file. When imported the node group will appear in the dimensional center of the Node Editor's Workspace area. Position offsets, connections, collapsed or expanded states and any node comments that existed when the nodes were exported will import back into your working environment.

This is an extremely useful feature allowing you to choose and load from maintained directory based libraries of various network segments or entire networks. ".node" files can be loaded and saved interchangeably between any of the three Node Editor windows for Displacements, Volumetric Lights, and Surfaces.

It is important to note that Destination node connections are not imported nor exported to or from a ".nodes" file.

## General Usage

Let's begin our exploration of the Node Editor, its interface and general use by performing some of the common tasks it was designed for. Pay close attention as we step through some of the simple operations used in common node based texturing. There will be a few hints and tips not covered elsewhere in this manual.

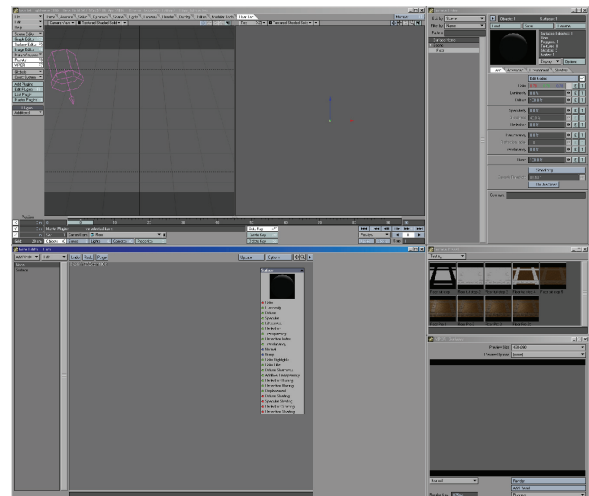
Start by setting up your interface so that you're working environment is suitable for texturing and shading object surfaces. Probably the first thing you will want to do is load the Studio Config files so that LightWaves' interface is full and well organized. This is accomplished by pressing [Alt+F10] and selecting Studio Production Style from the Presets pull down menu and clicking Done. Close any and all floating panels that may be open leaving only the Layout main program window displayed on your screen.

Next press [Ctrl+o] for the Load Scene requester and locate the scene file named "Floor Tutorial.lws". This should be in a folder named "ManualExamples" that was installed with your LightWave3D content.

After the scene has finished loading press [F5], [F7], [F8], and [F9] to open the Surface Editor, VIPER render preview, the Preset Shelf, and perform the initial render needed to initialize the VIPER display. Click on the Continue button in the Render Status panel when the renderer is finished to close the panel.

Next click on the Edit Nodes button in the Surface Editor panel to open the Node Editor and take a little time to arrange the five opened windows/panels in a configuration that you will be comfortable working with. Alternatively you may want to set up the window arrangement using the Window Config tool found in the Additional menu of the User Tab so that it can be saved and recalled at a later time. If you're not using the Studio Configs or do not see the Window Config tool simply press [Ctrl+Q] and select WindowConfigure from the Master Plugin's "Add Layout or Scene Master" pulldown menu.

Here is the way they are arranged on the machine where this tutorial is being created:



Just two more things to do to complete the environment



setup and we're ready to start our first texture. The first thing is to make sure that the check box next to the Edit Nodes button is indeed checked. Its state will determine whether or not nodes are applied to the surface. It is called the Enable Nodes Check Box and can be used to turn on or off the entire node network for any particular surface.

The second thing we need to do and is a good idea to perform any time you start a new surface is to create a new Preset Shelf Library. This is done by right clicking in the main area of the Surface Preset window and selecting Create Library from the Library submenu. Also very quickly select Normal instead of Draft for the quality settings in the VIPER display panel.

The object of this tutorial style instruction that we are about to undertake is primarily to familiarize you with the process of working with nodes so we will be focusing more on mouse and keyboard operations and the way the Node Editor handles itself under use rather than the actual texturing techniques involved in the steps we're about to perform.

Looking at the Node Editor window click on the Add Node button and select Grid2D from the 2D Textures submenu. If you are working with the Embedded Edit Panel Extension Area open the properties for the Grid2D node will be displayed on the right hand side of the Node Editor window. If it is not simply press enter to open its floating edit panel. The only thing we want to change in this panel is the Mortar Sharp value which we will set to 99.0% and the Axis which we will set to Y by clicking on the Y button in the same panel.

By clicking Undo at the top of the Node Editor window, we will notice that the Grid2D node is removed from the Workspace area - meaning that the Undo and Redo functions respect only Workspace area actions. This will be good to know as we work with the Node Editor interface.

Press Redo to get back the Grid2D node noticing that the 99% and the Y axis settings we entered prior to the undo operation are still in tact.

The Node Editor is setup with a logical flow construction from left to right. The colored dots and labels on the left hand side of any node are that nodes inputs. Any dots and labels located on the right hand side of a node are that nodes outputs. Inputs receive information from other nodes in the network while outputs send information.

Let's see how this works by connecting the Grid2D's "Color" output to the "Surface" destination nodes' "Color" input.

Place your mouse cursor over the red dot labeled "Color" on the right hand side of the Grid2D node. Now perform a click and drag operation toward the Surface destination Node's "Color" input dot. As you drag, an arrowheaded line will extend and follow your mouse cursor movements.

Notice that as your mouse moves into the body area of the Surface node that the arrowhead remains snapped to the input at the same latitude as your mouse cursor and not until your mouse cursor moves past the Surface node and emerges on the other side does the connection arrow unsnap and again begin following the mouse cursor. Position your mouse so that the arrow connects to the Surface nodes' Color input and release the mouse button to complete the drag operation.

As you release your mouse button with the arrow

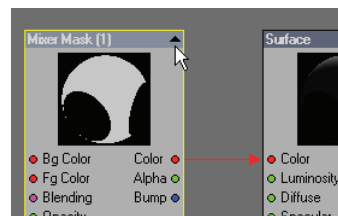
connected to the Color input of the Surface destination node you will notice that the VIPER display automatically renders the camera view of the floor object with the newly applied Grid2D procedural texture applied.

Double click anywhere in the VIPER preview display area to add a preset to your new surface library.

Click on the Grid2D node to select it and press [r]. Enter the name "Mixer Mask" into the Rename Node requester that appears and press [Enter]. Renaming a node has no affect on its function but is useful for keeping track of what nodes are doing and the role they play in a network. The Node Comment Entry Area

at the bottom of the Node Editor panel can also be used in this way. Whether either or both are used is a simple matter of user preference. With the node still selected click in the text entry area and enter some descriptive text such as: "Grid2D - Main Mask Mixer node defining the basic floor pattern" and press [Enter] when finished.

Collapse the newly named Mixer Mask node either by selecting it and pressing [C] or by clicking on the collapse toggle button in the nodes' title bar like so:



Click on the Add Node button and add the Parquet2D node from the 2D Textures submenu. Open its floating Edit Panel [ret] or use the embedded edit panel area to modify the following values:

Change U Tiles to 10, V Tiles to 10, and change the Axis to Y.

Now drag the Color output from the Parquet2D node to anywhere over the top of the Mixer Mask node and release the mouse button to finalize the drag operation.

Select BgColor from the menu and then expand the node either by pressing [E] or by clicking on the nodes collapse toggle button.

You will also notice that the VIPER preview has again automatically updated with the new surface information.

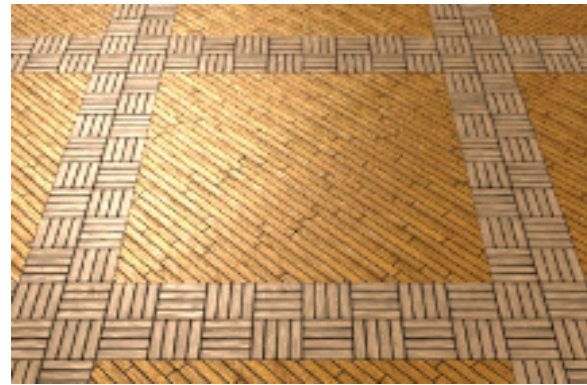
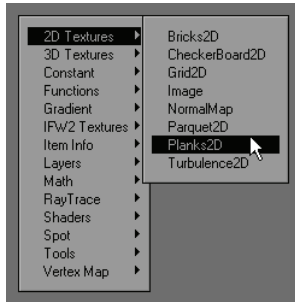
Again double click anywhere in the VIPER preview display area to add a second preset to the preset library we created earlier. You can come back to this point in the texture creation anytime you may need to simply by double clicking on the icon in the surface preset window.

Let's try that now by double clicking on the first icon you created, answer yes to the requested that appears, and look at the Node Editor Workspace area. Now click on the second preset you made, answer yes to the requester that appears, and we should be exactly back were we started prior to double clicking on any of the preset icons.

Next, hold down the control key while right clicking anywhere in the Node Editor Workspace area. You should see a popup menu that looks exactly like the Add Nodes menu at the top left of the Node Editor window.



Select Planks2D from the 2D Textures submenu to add that node to the Workspace area.



Connect the Color output from the Planks2D node to the Fg Color input of the Mixer Mask node and then edit the Planks2D node as we did the others before modifying the following values:

- Set Mortar Width to 10%.
- Set Mortar Sharp to 80%.
- Set both U Tiles and V Tiles to 40.
- Set the Axis to Y
- Set the Rotation Heading to 45° in the Rotation tab.

You should be seeing the lines of an uncolored wooden planks and parquet floor. The technique that is involved here is simple. The procedurally generated Alpha channel of the Grid2D texture is being used to mask and contain each of the two patterns added to its foreground and background color layers respectively.

Go ahead and double click on the VIPER preview area to add another preset to the preset shelf.

On your own go ahead and try to add some wood grain and color the surfaces appropriately.

Intermediate or advanced users may want to take a look at this example in its more developed form. To do so select "Floor\_Explore" from the pull down menu in the upper left hand corner of the Surface Preset window and double click on the various icons that were saved in steps as the texture was being created.

You may want to load the preset steps in the order they were created for more easy to follow explanation of how the final surface was created.

Here is an image of the floor surface in a fairly mature phase of its creation

In conclusion, we should now have a fairly basic understanding of how to optimize the LightWave3D environment and its windows for texturing and surfacing work. We saw the most common user operations for working with nodes in the Node Editor and explored several ways to achieve each one. Additionally users new to node based texture editing should have been able to glimpse some of the underlying power that node based systems offer and how much easier they are to use.



## Connection Types

There are 5 kinds of connections in the Node Editor. Each type (listed below) is color coded for your convenience. Color type connections are red, scalar types are green, vector types are blue, integer types are purple and functions are yellow. Normally you will want to keep the connections limited to only similar types but the Node Editor is very flexible and allows dissimilar types to be connected. This can be used to your advantage if you apply logic while constructing your networks.

It's useful and sometimes necessary to mind which types you are connecting when creating a network. There are a few rules of behavior that you may count on from the Node Editor and like all rules there are a few exceptions to mind as well. One very advantageous rule is that when connecting dissimilar types the output type will be automatically converted to the input type that you are connecting it to, in the most intelligent way possible. There are only three exceptions to this and they are:

1. The Normal input in the destination Surface node requires the incoming type be a vector.
2. The Bump input in the destination Surface node requires the incoming type be a vector.
3. Connecting another type of output to a Function input will not convert it to a function type. The connection will be made and some of the information from the non-function output may be used but the results will be unpredictable. Functions are the only type of connection that are actually bidirectional. This bidirectionality is transparent to the user but Function nodes receive information from the node they are modifying, alter the information in some way and then send the results back to the node they are connecting to. A good general rule of thumb to remember about function type nodes is just to not intermix them with dissimilar types. *Keep your Functions in the yellow and all will be mellow.*

Use the table at the end of this chapter for quick look up of automatic conversion between same and dissimilar connection types.

Here is a description of the five kinds of connections found in the Node Editor and some more of the rules that apply to each one in greater detail.

## Color Type Connections

Color outputs and inputs are designated by a red dot next to the connection name. Color is a composite type connection containing three channels of information.

### Input:

Color inputs receive incoming color connections as three channels red (R), green (G), and blue (B) from other nodes in the network you are currently working with. Most of the time you'll want to connect colors to colors unless you have a very specific task in mind and have read this documentation thoroughly fully understanding the consequences and conversions that take place when making connections between dissimilar connection types.

Connecting non-color output from other nodes to a Color input will automatically copy the incoming values to each color channel red, green, and blue. The color of the line connecting the two dissimilar types will change over the lines length from whatever color designates the output selected to red.

If the incoming connection is a vector type output then XYZ or HPB etc., will supply the values for the RGB respectively. This means that if the incoming vector type is a Position for example, then the X value of that vector will be used to define the red channel of the color. Y to green, and Z to blue and so on in a similar way for HPB if the vector being used is a rotation and so on.

The vast majority (almost all) of connections in the Node Editor are evaluated per spot therefore if the incoming color connection has a variance in the shader or texture values that is affecting the color channel (i.e. a pattern of some kind) then that varied color value will be supplied to the color input it is connecting to. Simply put, if you connect the color output of a red and green marble texture to a color input then that input will receive a red and green colored marble pattern - evaluated on a spot by spot basis.

### Output:

Color outputs always supply three channel data evaluated per spot, in the format red, green, and blue (RGB) regardless of what other inputs are connected to the node or what other settings you may have set for your scene. Even if the incoming data for the node's colors are scalar values and/or you have selected grey values via the color picker panel, the color output will still be outputting three channels of data.

See the Connection Table at the end of this chapter for a complete list of connections and conversions.





## Scalar Type Connections

Scalar outputs and inputs are designated by a green dot next to the connection name. In LightWave 3D's Node Editor, a scalar is a floating point number or quantity usually characterized by a single value which doesn't involve the concept of direction. The term scalar is used in contrast to entities that are "composites" of many values, like vector (XYZ, HPB, etc.), color (RGB), and etc.

Some examples of scalar values are:

3.14159265, 13, 0.00001, -5.5, 2, and 69.666

**Input:** Primarily, connections to a scalar input will be scalar outputs from other nodes in the network you're working with. Most of the time you'll want to connect scalars to scalars unless you have a very specific task in mind and have read this documentation thoroughly fully understanding the consequences and conversions that take place when making dissimilar connections.

Connecting non-scalar output values to a scalar input will automatically convert it to scalar type. The color of the line connecting the two dissimilar types will change over the lines length from whatever color designates the output selected to green.

If a component (vector or color) type output is being connected to the scalar input then one of several things may happen as follows:

If it is a color type output that is being connected to the scalar input then the CCIR 601 luminance values for that color are derived from the RGB values and used as the scalar input.

If it is a vector type output that is being connected to the scalar input then only the first component of the vector is used as the scalar value. For example if the vector being used is a position then only X will be supplied to the scalar input and Y and Z will be ignored. Likewise if the vector being used is a rotation then only the H (heading) will be used and the P (pitch) and B (bank) channels will be ignored.

Again however, these conversion processes are automatic and transparent to the user.

**Output:**

Scalar outputs from any given node will always supply scalar values regardless of what other inputs are connected to the node. Please see above for examples of scalar values.

See the Connection Table at the end of this chapter for a complete list of connections and conversions.

## Integer Type Connections

Integer outputs and inputs are designated by a purple dot next to the connection name.

Integers in the Node Editor can be used for many things but most often they are designated for the purpose of specifying predefined list member items such as any of the Blending modes described below in the next section on Blending Modes (see the Blending Modes table further down).

An integer is a non-floating point natural number. Some examples of integers are:

3, 13, 0, -5, 69 and 666

**Input:**

Primarily connections to an integer input will be integer outputs from other nodes in the network you're working with. Most of the time you'll want to connect integers to integers unless you have a very specific task in mind and have read this documentation thoroughly thus fully understanding the consequences and conversions that take place when making dissimilar connections.

Connecting non-integer outputs to an integer input will automatically convert it to integer type using standard rounding. The color of the line connecting the two dissimilar types will change over the lines length from whatever color designates the output selected to purple.

If a component type output is being connected to the integer input then one of several things may happen as follows:

If it is a color type output that is being connected to the integer input then the CCIR 601 luminance values for that color are derived from the RGB values. Those luminance values are then rounded and used as the integer input. Since internal luminance values in the Node Editor are floating point values which most often range from 0.0 (black) to 1.0 (white) the rounding that takes place during this conversion would have the effect of reducing the luminance information to either a value of zero or one with no grey tones in between. Consider that the luminance value 0.5 (50% grey) when rounded to an integer value would become a value of 1 (white) and a grey tone just slightly below 50% grey (0.4) when rounded to an integer value would be 0 (black).

If it is a vector type output that is being connected to the integer input then only the first component of the vector is used after rounding it, as the integer value. For example if the vector being used is a rotation then only the rounded H (heading) value will be supplied as the integer input and P (pitch) and B (bank) will be ignored. Likewise if the vector being used is a scale or position then only the rounded X value will be used and the Y and Z channels will be ignored.

These conversion processes however, are automatic and transparent to the user.

**Output:**

Integer outputs from any given node will always supply integer values regardless of what other inputs are connected to the node. Please see above for examples of integer values.



## Vector Type Connections

Vector inputs and outputs are designated by a blue dot next to the connection name.

Vectors are composite type values meaning they are comprised of more than one component or "channel". Some examples of vectors in the Node Editor are Position, Scale, and Rotation. Notice that each one of these attributes contain more than one value. For example position is comprised of X, Y and Z. Scale is also comprised of X, Y, and Z and rotation is comprised of H, P and B - Heading, Pitch, and Bank respectively.

### Input:

Generally, connections to vector type inputs will be vector type outputs from other nodes in the network you're working with. Most of the time you'll want to connect vectors to vectors unless you have a very specific task in mind and have read this documentation thoroughly thus fully understanding the consequences and conversions that take place when making dissimilar connections. You will never want to connect anything other than a vector type output to a vector type input that is labeled "Normal" next to the blue dot designation unless you know ahead of time why. The case for needing to make such kinds of connections to a Normal input is so rare that in fact the destination Surface node disallows non-vector connections to its Bump and Normal vector inputs. The general rule of thumb here is that all Normal inputs should only be receiving their data from other Normal type outputs. You may remember it by learning: "Normals in like only Normals out." where the inverse of that phrase may not always be true.

Connecting non-vector output types to a vector input will automatically convert it to vector type with various limitations. The connection will be made except where noted, and the color of the line connecting the two dissimilar types will change over the lines length from whatever color designates the output selected, to blue.

Color type outputs are a kind of vector although undesignated as such and for good reason. It can be quite confusing to discuss colors as vectors. However there are times when the relationships are straightforward enough to be immediately useful.

When connecting a color output to a vector input each channel of the color output (R, G, and B) will be used to define the first, second, and third components of the vector respectively.

If a non-component type (scalar or integer) output is being connected to the vector input then the value of the scalar or integer is copied into all of the components of the vector that it is being connected to. For example connecting an Alpha output from any of the texture nodes to any of the texture nodes input connections labeled Scale would copy the Alpha value to all three of the components of the Scale vector. If for example, the Alpha value was 1.0 the recipient textures Scale value would be 1.0 for X, Y, and Z.

Once again these conversion processes are automatic and transparent to the user.

### Output:

Vector outputs from any given node will always supply vector component values regardless of what other inputs are connected

to the node.



## Function Type Connections

Function type connections are special 2-way connections. Functions offer a way of transforming a texture or shader value based on the functions' graph.

### Input:

Function inputs should only be connected to from other Function type nodes in the network you are working with.

Always connect function to functions.

Connecting non-function outputs to Function inputs will make the connection and the color of the line connecting the two dissimilar types will change over the lines length from whatever color designates the output selected, to yellow. However the results of dissimilar connections with function nodes are not guaranteed to produce any predictable results.

When a function output is connected to a function input data (usually texture value data) from the node being connected to is sent to the connecting node where it is then transformed and sent back to the recipient node. The transformations that occur depend entirely upon the kind of Function node that is being used and the settings entered for its graph.

### Output:

Function nodes specifically utilize the function connection to transform values of the nodes they are connecting to. Typically this will be shaders and textures and in these cases the shader or texture sends the shader or texture value to the connected function node. The connected function node then transforms these values based on the function graph, and returns the transformed values to the recipient node.

Function connections are specialized in this way allowing for bidirectional communication between nodes.

Once again, the output of a Function type connection should always be connected to the input of another Function type connection.

## Blending Modes

Value	Blending Mode
0	Normal
1	Additive
2	Subtractive
3	Multiply
4	Screen
5	Darken
6	Lighten
7	Difference
8	Negative
9	Color Dodge
10	Color Burn
11	Red
12	Green
13	Blue

Please see the chapter on the Surface Editor Procedurals for a detailed description of these blending types. The blending types unique to the Node Editor are Darken, Lighten, Color Dodge, Color Burn, Red, Green, and Blue. The Blending type determines how the FG and BG colors are blended.

All of the blending modes in the Node Editor are based on their Photoshop equivalents. Here is a description of each one not defined elsewhere in the LightWave 3D manual:

### Darken:

Darken looks at the foreground and background colors and chooses the darker one, whichever it is. Whichever is darker wins. Which of the two is used will vary across the image depending on which is darker at each spot as it is evaluated.

### Lighten:

Lighten is the reverse of darken. It evaluates the foreground and background colors and replaces the darker ones with the lighter ones.

### Color Dodge:

Color Dodge is a kind of inverted multiply mode. Dodging is a dark room technique commonly used by photographers and photo developers. This analog equivalent technique is achieved by holding a small piece of cut paper taped to the end of a thin wire or stick, in between the projected light and the photo paper causing some areas of the photograph to be underexposed. The same thing is going on here but the foreground pattern is being used as the dodge mask instead of a cut to shape piece of paper, to underexpose those areas. Remember that the more underexposed an area is the lighter it is and a completely unexposed sheet of photo paper is all white.

### Color Burn:

Color Burn is basically the same thing in reverse. Burning is also a darkroom technique where (in its simplest form) a large piece of paper with holes cut in the middle is held between the projected light and the photo paper after exposing the image initially on to the paper. This has the effect of overexposing or burning in some areas of the image. Again it is the background color values and patterns that are being used in place of that cut sheet of paper just mentioned, to burn in the image. Remember that the more overexposed an area is the darker it is and a completely overexposed sheet of photo paper is all black.

**Red:**

Colors as seen on your monitor are comprised of three components red, green and blue. Each of these primary colors can be talked about or represented by considering them as separate channels. Usually to form any specific color these three channels are mixed additively.

In these next three blending modes we are able to use each unique channel individually.

Selecting Red as the blending mode replaces the blue and green of the FG color channels with the blue and green channels contained in the BG Color. Only the foreground's red channel will be considered but the green and blue channels will be taken from the background color only. If for example, your background color was 100% blue and your foreground color was 100% red - selecting Red as the blending mode would result in purple output.

**Green:**

Selecting Green as the blending mode replaces the blue and red channels of the FG color with the blue and red channels from the BG Color. Only the foreground's green channel will be considered but the red and blue channels will be taken from the background color only.

**Blue:**

Selecting Blue as the blending mode replaces the red and green channels of the FG color with the red and green channels contained in the BG Color. Only the foreground's blue channel will be considered but the red and green channels will be taken from the background color only.

**Opacity:**

Opacity is the amount that each of these blending modes mentioned above, is applied. You can think of opacity as the strength of the blend effect.

**Invert:**

The Invert toggle button inverts the scalar Alpha values used to blend the two colors; background and foreground. Where applicable it will also invert the bump gradient so that not only are the background and foreground colors are swapped but the bump and alpha outputs as well are inserted.

## About "Usage Examples"

For almost every 2D and 3D texture node covered in this manual there is a Usage Example section included as kind of a mini tutorial. The first few will be extremely simple and very straightforward however as we progress through the nodes some intermediate texturing and surfacing concepts will be added here and there.

All the nodes and textures included in LightWave 3D are multipurpose. Some textures are useful for things that you would not immediately recognize them as being useful for. For example Bricks2D can be used to make strange looking spider webs and a various assortment of other similar patterns. Mostly throughout this manual we will be choosing very practical and common applications for each texture example. Texturing and shading examples will be applied to one of three kinds of objects; a 1 meter in diameter sub-divided sphere, a 1 meter sub-divided cube with rounded corners, or a 1 meter sub-divided cube with stubs attached to each of the six sides.

On a special note please take into consideration is that each shading model or texture used in the examples throughout this manual can be applied directly to other objects in scenes that you are creating for your own custom projects. However, some of the values used may need to be adjusted in order to look correct within the lighting, scale, and environment properties of the scene you are working with. The properties and values applied to a texture or a shader for use in the Default Scene environment and for the purpose of displaying on a printed page may be quite different than what is needed when adapting them to your own projects.

It's good practice to work with the VIPER window [F7] open at all times while creating networks with the Node Editor. It is also very beneficial to have the Surface Presets shelf [F8] open during any surfacing work. The two panels VIPER and the Presets shelf, work together to offer the user a convenient interactive work history during the process of constructing texturing or shading Networks. Double clicking on a VIPER displayed image will add an entry in the Presets shelf with a scale version of that image as the icon for that preset. Conversely, double clicking on an icon in the Presets shelf will load and apply all of the surface settings saved when you double clicked on the VIPER display.



NOTE: A Preview Render [F9], must be performed to initialize the VIPER display or update it after any geometry or keyframe changes. Additionally the Surface Editor must remain open (even if minimized) for VIPER to continue to update automatically as you work with surfaces in either the Surface Editor or the Node Editor. Please see the sections elsewhere in this manual for detailed descriptions of how to use both VIPER and the Presets Shelf.



---

## 2D Textures

---

2D textures consist of images or procedurals that can be mapped to the object surface during the shading computation. An algorithm is used for “wrapping” the texture around the object’s surface so that the pattern curves and distorts realistically. The advantage of using 2D textures is that in using the projection methods available to them the texture patterns can conform easily to basic geometrical shapes. Additionally textures in general add apparent detail and so do not have to be modeled in geometry. A common use of a 2D texture might be to apply an image as a decal or label on a three-dimensional bottle or other object. 2D textures can also be used as transparency, reflection, refraction, normal, and displacement maps etc.





## Bricks2D

### Description

Bricks 2D is a simple brick like 2D texture useful for very many patterns and effects that one would not normally expect just by considering its name. From broken glass and strange spider webs to more in line with what its name implies; such as roughly shaped cobblestone roads and brick walls.

The Bricks2D texture has axial projection methods available such as spherical, cubic, planar, front, cylindrical and UV.

### Inputs

#### Bg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a background color using the controls found in the Edit Panel for this node.

Specifies the grout or mortar color.

#### Fg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a foreground color using the controls found in the Edit Panel for this node.

Specifies the brick face color.

#### Blending (type: integer):

Specifies one of the blending types (0~13) found in the blending type list at the beginning of this chapter.

Can receive input from other nodes in the network or can be specified to by selecting a blending mode from the Blending pull-down menu in the Edit Panel for the node.

For general use specifying the blending mode by using the Blending pull-down menu in the Edit Panel will probably be the most desirable method of use. However, for larger networks where control of the blending modes of many texture nodes simultaneously is needed or when conditional blending is desired then using a node connection to control this value may be advantageous.

#### Opacity (type: scalar):

Opacity is the amount that the selected blending mode is applied. You can think of opacity as the strength of the blend effect between the BG Color and the FG Color.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### U Tiles (type: integer):

Specifies the number of tiles in the U dimension.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### V Tiles (type: integer):

Specifies the number of tiles in the V dimension.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Mortar Width (type: scalar):

Specifies the width of the mortar between the brick faces. Increasing this value will also reduce the face of the bricks in such a way that the pattern will repeat correctly the number of U and V tiles within the defined scale of the texture.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Mortar Sharp (type: scalar):

Defines how steep the falloff angle is between the brick faces and the bottom trough of the mortar area.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Mortar Shift (type: scalar):

Mortar shift specifies the offset of the U directional mortar areas. When creating a standard red brick surface you can think of this value as determining the stagger amount of the bricks rows.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### U Offset (type: scalar):

This specifies the offset value of the U tiles. This is a little bit like Position but only for the U value of the UV tiles.

Can receive patterns and numerical inputs from other nodes in the network.

#### V Offset (type: scalar):

This specifies the offset value of the V tiles. This is a little bit like Position but only for the V value of the UV tiles.

Can receive patterns and numerical inputs from other nodes in the network.



## Function (type: function):

Connecting a function node to this input will modify the texture value of this texture. See the manual section on Function nodes for a description of how texture values can be modified.

Can receive inputs from other function nodes in the network only.

## Bump (type: vector):

Specifies a vector direction for modifying the surface normal. This is surface normal direction information and affects the way the surface is shaded. Care should be taken when connecting to this input. Connecting dissimilar types or non directional vectors may cause the surface to shade wrongly.

Can receive input from other nodes in the network.

## Bump Amplitude (type: scalar):

Specifies the bump height or “amplitude” of the Bump directional vectors.

Using Alpha or luminance (connecting Color) values from other textures in the network along with their respective bumps can have the effect of blending the bump textures together.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Since Bump Amplitude is a shading feature and not a displacement coordinate system value it is specified as a percentage. Specifically as a percentage of the texture value.

## Scale (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Scale receives its X, Y, and Z scale values either by entering them into the Scale tab in the Edit Panel or by connecting an input to the Scale connection on the node in the Workspace.

Scale is the scale of the texture pattern that will be applied to your object surface.

If the World Coordinates checkbox is checked in the edit panel then Scale defines the scale of the texture in relation to the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, and Z respectively.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures scale is derived from the selected Reference Object keyframed Scale values.

Connections made in the workspace area will override any values entered by the user into the edit panel.

## Position (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Position receives its X, Y, and Z coordinate information either by entering it into the Position tab in the Edit Panel or by connecting an input to the Position connection on the node in the Workspace Area. Position defines the center coordinate of the texture. By default these are offset values of the objects coordinates also sometimes referred to as the objects pivot point.

If the World Coordinates checkbox is checked in the Edit Panel then Position defines the center of the texture as offset values from the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, Z format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures position is derived from the selected Reference Objects keyframed Position values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Rotation (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Rotation receives H (Heading), P (Pitch), and B (Bank) rotational information either by entering degrees into the Rotation tab in the Edit Panel or as radians by connecting an input to the Rotation connection on the node in the Workspace Area.

Rotation defines the axial rotation around the textures center point. By default these are offset values of the objects rotational settings.

If the World Coordinates checkbox is checked in the Edit Panel then Rotation values define the texture rotation as offset values from the world coordinates of your scene which should be fixed. You can rely on your world coordinate rotation values to always be 0,0,0, in H, P, B format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures rotation is derived from the selected Reference Objects keyframed Rotation values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.



## Outputs

### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

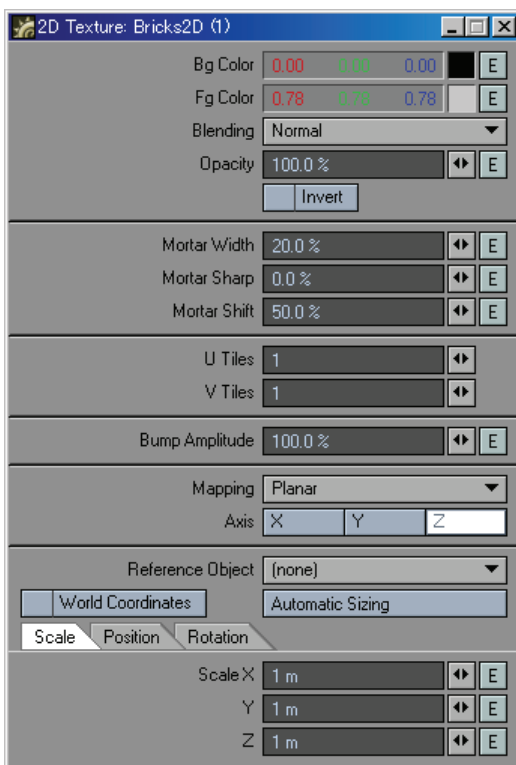
### Alpha (type: scalar):

Outputs single channel alpha (grey) information evaluated on a per spot basis.

### Bump (type: vector):

Outputs component vector direction information the length of which are the product of the bump amplitude value.

## Edit Panel



Here the Mapping projection types and Axis are not offered as node connections in the Workspace area. Likewise Reference Object, World Coordinates and Automatic Sizing are features exclusive to this nodes Edit Panel. Please see the explanation for these items in the chapter that covers texturing with the Surface Editor.

## Usage Example

In this example we will be using Bricks2D to apply a brick pattern to a simple sphere object. Example objects and presets are available on the content disk that came with LightWave 3D for your convenience.

Here is the result of the network we're about to create:



After you have loaded the "Manual\_NodeEditor\_Usage\_Example\_Environment.lws" scene file found in the Scenes\Manual\_Examples\ subfolder, click on the Edit Nodes button in the Surface Editor panel, to open the Node Editor. Click the Add Node button and find Bricks2D in the 2D Textures submenu to add it to the Node Editor Workspace.

It's good practice to work with the VIPER window [F7], open at all times while creating networks with the Node Editor. It is also very beneficial to have the Surface Presets shelf [F8], open during any surfacing work. The two panels VIPER and the Presets shelf, work together to offer the user a convenient interactive work history during the process of constructing texture or shading Networks. Double clicking on a VIPER displayed image will add an entry in the Presets shelf with the image displayed in VIPER as the icon for that preset. Conversely, dubbed clicking on an icon in the Presets shelf will load and apply all of the surface settings saved when you double clicked on the VIPER display.

One of the very first things you will want to do when working with a texture node that is new to you is connect its Color output to the Surface destination nodes Color input to get an idea what it looks like with its default settings. Next you will want to double click on the node and modify the values in the Scale tab inside the Edit Panel that appears, in order to scale the texture appropriately for the geometry you are working with.

Sometimes and especially with 2D Textures the process of fitting or scaling a texture to your geometry is facilitated by setting the number of U and V tiles, selecting a Mapping projection type and choosing an Axis (X, Y, or Z) for that Mapping projection method.

The next step in new node discovery, will be to connect up any other outputs that may exist in a similar fashion one at a time. Often what you will discover during this process may surprise you.

Disconnect the Bricks2D Color output from the Surface Color input and connect the Alpha output to the Surface Color input



by simply dragging the connection from the Alpha over the top of the Color input connection that already exists. You are now looking at the Alpha channel for the Bricks2D procedural texture.

After you have had a chance to visualize this output disconnect the Alpha output by dragging it off of the Surface nodes Color connection and releasing it.

The Bricks2D texture node only has three outputs so let's connect the third and final output so we can get an idea of what it does. Connect the Bump output from the Bricks2D node to the Surface destination nodes Bump input.

Disconnect it when you have finished viewing the results and you have just completed a discovery process that it would behoove you to perform on every new shading or texturing node you encounter.

Let's move on to producing the white brick texture shown in the image above. Double click on the Bricks2D node to bring up its Edit Panel. The settings used in the example are:

Mortar Width: 13%  
Mortar Sharp: 25%  
U Tiles: 10  
V Tiles: 14  
Mapping: Cylindrical  
Axis: Z

With all other settings at their default values connect the Color output from the Bricks2D node to the color input of the Surface destination node. Connect the Alpha output from the Bricks2D node to the Specular input of the surface destination. Finally, connect the Bump output from the Bricks2D node to the Bump input of the Surface destination node.

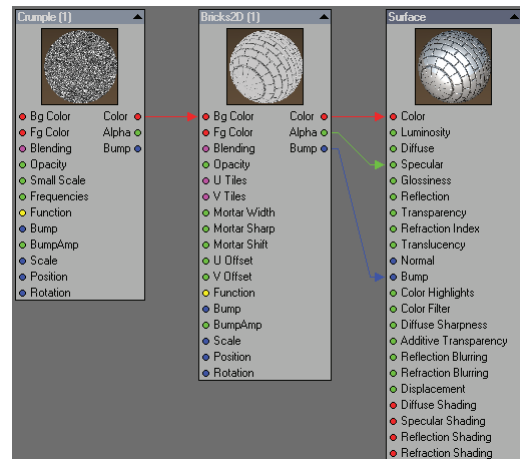
That looks OK but we want the mortar area of the bricks to look rough and more like real brick mortar. In the Bricks2D node the background color Bg Color, defines the color of the mortar area. We can connect a texture to the Bg Color input of the Bricks2D node in order to place a texture pattern in the mortar area of the brick pattern. Let's try this by adding a Crumple node.

Click on the Add Node button and then find and select the Crumple texture node from the 3D Textures submenu. Double click on the Crumple node you just added and modify the values in the Scale tab.

We want to scale the Crumple texture down to about the size of the pattern roughness of real world mortar in relation to our brick faces. In relation to the size of the bricks currently being applied to this one meter in diameter sphere that this example uses, three centimeters (3cm) for each X, Y, and Z, scale value were applied.

This example uses the default Crumple node foreground and background colors which you may want to change in order to tone down the effect or to more closely resemble the colors of actual mortar.

When finished connect the Color output from the Crumple node to the background color (Bg Color) of the Bricks2D node. This completes this example and if you did everything according to instruction your node network should now look like this:



Additionally, your VIPER preview should be looking very much like the image we saw at the beginning of this example.



## CheckerBoard2D

### Description

Procedurally generated checkerboard pattern with axial projection methods such as spherical, cubic, planar, front, cylindrical and UV.

### Inputs

#### Bg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the background color layer

#### Fg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the foreground color layer

#### Blending (type: integer):

Specifies one of the blending types (0~13) found in the blending type list at the beginning of this chapter.

Can receive input from other nodes in the network or can be specified to by selecting a blending mode from the Blending pull-down menu in the Edit Panel for the node.

For general use, specifying the blending mode by using the Blending pull-down menu in the Edit Panel will probably be the most desirable method of use. However, for larger networks where control of the blending modes of many texture nodes simultaneously is needed or when conditional blending is desired then using a node connection to control this value may be advantageous.

#### Opacity (type: scalar):

Opacity is the amount that the selected blending mode is applied. You can think of opacity as the strength of the blend effect between the BG Color and the FG Color.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### U Tiles (type: integer):

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

Specifies the number of tiles in the U dimension.

#### V Tiles (type: integer):

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

Specifies the number of tiles in the V dimension.

#### U Offset (type: scalar):

This specifies the offset value of the U tiles. This is a little bit like Position but only for the U value of the UV tiles.

Can receive patterns and numerical inputs from other nodes in the network.

#### V Offset (type: scalar):

This specifies the offset value of the V tiles. This is a little bit like Position but only for the V value of the UV tiles.

Can receive patterns and numerical inputs from other nodes in the network.

#### Scale (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Scale receives its X, Y, Z, scale values either by entering them into the Scale tab in the Edit Panel or by connecting an input to Scale on the node in the Workspace.

Scale is the scale of the texture pattern that will be applied to your object surface.

If the World Coordinates checkbox is checked in the edit panel then Scale defines the scale of the texture in relation to the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, and Z respectively.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures scale is derived from the selected Reference Object keyframed Scale values.

Connections made in the workspace area will override any values entered by the user into the edit panel.

#### Position (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Position receives its X, Y, Z, coordinate information either by entering it into the Position tab in the Edit Panel or by connecting an input to the Position connection on the node in the Workspace Area.

Position defines the center coordinate of the texture. By default these are offset values of the objects coordinates also sometimes referred to as the objects pivot point.

If the World Coordinates checkbox is checked in the Edit Panel





then Position defines the center of the texture as offset values from the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, Z format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures position is derived from the selected Reference Objects keyframed Position values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Rotation (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Rotation receives H (Heading), P (Pitch), and B (Bank) rotational information either by entering degrees into the Rotation tab in the Edit Panel or as radians by connecting an input to the Rotation connection on the node in the Workspace Area.

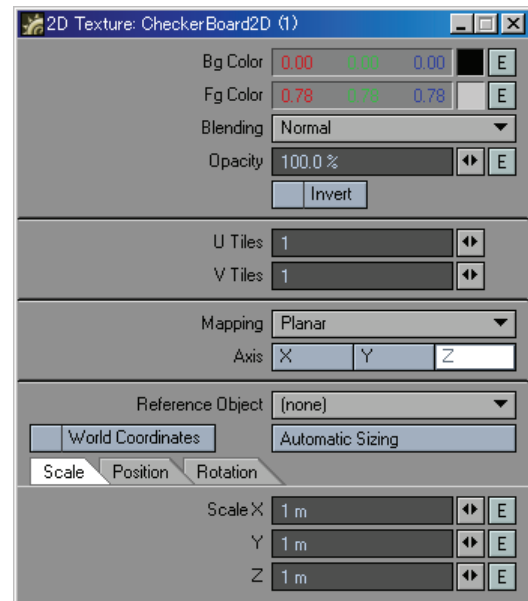
Rotation defines the axial rotation around the textures center point. By default these are offset values of the objects a rotational settings.

If the World Coordinates checkbox is checked in the Edit Panel then Rotation values define the texture rotation as offset values from the world coordinates of your scene which should be fixed. You can rely on your world coordinate rotation values to always be 0,0,0, in H, P, B format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures rotation is derived from the selected Reference Objects keyframed Rotation values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Edit Panel



Here the Mapping projection type and Axis are not offered as node connections in the Workspace area. Likewise Reference Object, World Coordinates and Automatic Sizing are features exclusive to this nodes Edit Panel. Please see the explanation for these items in the chapter that covers texturing with the Surface Editor.



NOTE: Radians are used as the unit of measure when making connections between nodes. Angles in degrees are used when user input is specifying the value. This is incredibly advantageous as you will discover as you work more and more with nodes.

## Outputs

### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

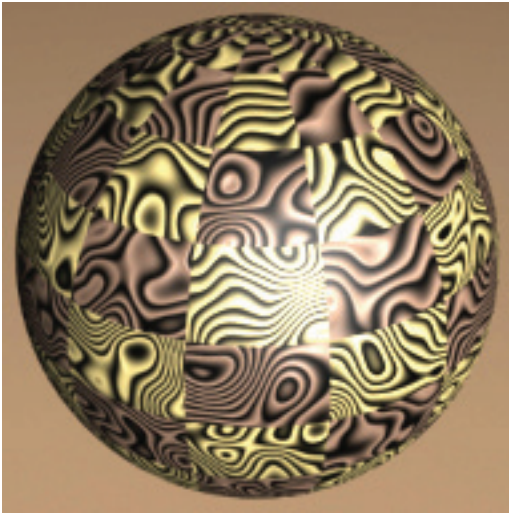
### Alpha (type: scalar):

Outputs single channel alpha (grey) information evaluated on a per spot basis.



## Usage Example

Here is the final render for the checkerboard example.



Here we can see the node network displayed.



We are using the checkerboard texture in two very specific and distinct ways. Often while creating networks you'll notice that nodes can function for more than one purpose within the network. Here the Checkerboard2D is being used to colorize the Wood2 texture and at the same time rotate the wood texture in such a way that each "block" looks like a different wood texture.

Using the checkerboard to rotate the blocks is the most interesting aspect of this example.

Remember that radians are used for units of rotation for connected inputs in the workspace area. Since our checkerboard Alpha map is solid white (1) and solid black (0) only, then only two rotational values are ever interpreted from the Alpha channel when we plug it in to the Rotation input of the Wood2 node.

Since one radian is equal to about  $57.17^\circ$  and zero radians are equal to  $0^\circ$  then wherever the incoming Alpha values are 1 the wood texture will be rotated approximately  $57.17^\circ$  on all three axis H, P, and B. And wherever the Alpha value is 0 the wood texture will be rotated  $0^\circ$ .

## Grid2D

### Description

Procedurally generated grid pattern. This grid pattern can vary the thickness of the lines that form the grid as well as define a soft edge falloff from line to center in user specified amounts. It has axial projection methods available such as spherical, cubic, planar, front, cylindrical and UV.

Grid2D is also extremely useful for visualizing results from the output of other nodes and network segments.

### Inputs

#### Bg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the background color layer.

#### Fg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the foreground color layer.

#### Blending (type: integer):

Specifies one of the blending types (0~13) found in the blending type list at the beginning of this chapter.

Can receive input from other nodes in the network or can be specified to by selecting a blending mode from the Blending pull-down menu in the Edit Panel for the node.

For general use specifying the blending mode by using the Blending pull-down menu in the Edit Panel will probably be the most desirable method of use. However, for larger networks where control of the blending modes of many texture nodes simultaneously is needed or when conditional blending is desired then using a node connection to control this value may be advantageous.

#### Opacity (type: scalar):

Opacity is the amount that the selected blending mode is applied. You can think of opacity as the strength of the blend effect between the BG Color and the FG Color.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.



## U Tiles (type: integer):

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

Specifies the number of tiles in the U dimension.

## V Tiles (type: integer):

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

Specifies the number of tiles in the V dimension.

## Mortar Width (type: scalar):

Specifies the width of the mortar between the grid squares or "faces". Increasing this value will also reduce the size of the face area of the grid in such a way that the pattern will repeat correctly the number of U and V tiles within the defined scale of the texture.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

## Mortar Sharp (type: scalar):

Defines how steep the falloff angle is between the grid faces and the bottom trough of the mortar area.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

## U Offset (type: scalar):

This specifies the offset value of the U tiles. This is a little bit like Position but only for the U value of the UV tiles.

Can receive patterns and numerical inputs from other nodes in the network.

## V Offset (type: scalar):

This specifies the offset value of the V tiles. This is a little bit like Position but only for the V value of the UV tiles.

Can receive patterns and numerical inputs from other nodes in the network.

## Function (type: function):

Connecting a function node to this input will transform the texture value of this texture based on the graph defined by the function thereby extending the range of this nodes producible patterns. See the manual section on Function nodes for a description of how texture values can be transformed.

Can receive inputs from other function nodes in the network only.

## Bump (type: vector):

Specifies a vector direction for modifying the surface normal. This is surface normal direction information and affects the way the surface is shaded. Care should be taken when connecting to this input. Connecting dissimilar types or non directional vectors may cause the surface to shade wrongly.

Can receive input from other nodes in the network only.

## Bump Amplitude (type: scalar):

Specifies the bump height or "amplitude" of the Bump directional vectors.

Using Alpha or luminance (connecting Color) values from other textures in the network along with their respective bumps can have the effect of blending the bump textures together.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Since Bump Amplitude is a shading feature and not a displacement coordinate system value it is specified as a percentage. Specifically as a percentage of the texture value.

## Scale (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Scale receives its X, Y, Z, scale values either by entering them into the Scale tab in the Edit Panel or by connecting an input to Scale on the node in the Workspace.

Scale is the scale of the texture pattern that will be applied to your object surface.

If the World Coordinates checkbox is checked in the edit panel then Scale defines the scale of the texture in relation to the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, and Z respectively.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures scale is derived from the selected Reference Object keyframed Scale values.

Connections made in the workspace area will override any values entered by the user into the edit panel.



## Position (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Position receives its X, Y, Z, coordinate information either by entering it into the Position tab in the Edit Panel or by connecting an input to the Position connection on the node in the Workspace Area.

Position defines the center coordinate of the texture. By default these are offset values of the objects coordinates also sometimes referred to as the objects pivot point.

If the World Coordinates checkbox is checked in the Edit Panel then Position defines the center of the texture as offset values from the world coordinates of your scene. Also known as world coordinate 0,0,0, in X, Y, Z format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures position is derived from the selected Reference Objects keyframed Position values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Rotation (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Rotation receives H (Heading), P (Pitch), and B (Bank) rotational information either by entering degrees into the Rotation tab in the Edit Panel or as radians by connecting an input to the Rotation connection on the node in the Workspace Area.

Rotation defines the axial rotation around the textures center point. By default these are offset values of the objects a rotational settings.

If the World Coordinates checkbox is checked in the Edit Panel then Rotation values define the texture rotation as offset values from the world coordinates of your scene which should be fixed. You can rely on your world coordinate rotation values to always be 0,0,0, in H, P, B format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures rotation is derived from the selected Reference Objects keyframed Rotation values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Outputs

### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

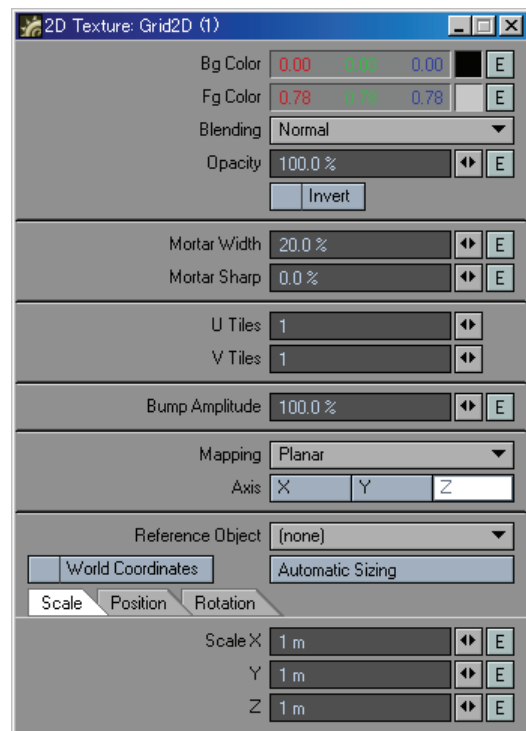
### Alpha (type: scalar):

Outputs single channel alpha (grey) information evaluated on a per spot basis.

### Bump (type: vector):

Outputs component vector direction information the length of which are the product of the bump amplitude value.

## Edit Panel



Here the Mapping projection type and Axis are not offered as node connections in the Workspace area. Likewise Reference Object, World Coordinates and Automatic Sizing are features exclusive to this nodes Edit Panel. Please see the explanation for these items in the chapter that covers texturing with the Surface Editor.

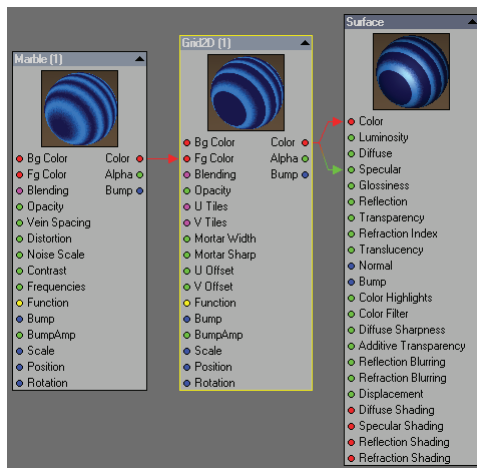


## Usage Example

Here is the rendered image of this example:



Here is what the node network looks like:



In this simple example the Grid2D node is being used to mask off or “overwrite” half of the grainy gradient bands produced by another texture; Marble. This technique is achieved merely from scaling and positioning the grid in alignment to the texture that it is masking.

Here the color output from the Grid2D texture is connected to the Specular input on the surface destination node. Remember, that when a color is connected to a scalar input such as Specular, that the color is converted into single channel luminance values.

See the “Connections” section in the first part of the Node Editor chapter for further explanation on connections and their conversions.

See the Grid2D example preset or object file in your content folder for a breakdown of the settings used to achieve this surface.

## Image

### Purpose

To make use of loaded images within a network or as freestanding connections directly connected to the destination node.

### Description

The description for this particular node could fill volumes and in fact the use of images as applied to shading, texturing and lighting, has! This node is probably the single most useful and versatile node. The image node will allow you to load images with or without Alpha channels, map them directly onto polygons, and use various derived values from the image itself to control other nodes in the network.

The Image node has axial projection methods available such as spherical, cubic, planar, front, cylindrical and UV.



NOTE: It is worth mentioning here that while either images or procedurals by themselves can create stunning results, it is extremely advantageous to use procedurals and images together! You may find that when creating textures for photo realistic or semi photo realistic rendering, that procedurals can complement images and vice versa, in a way that far outweighs the use of either alone by themselves.

### Inputs

#### Bg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the background color layer. For this node the foreground layer is being specified by the image you select in the node Edit Panel.

#### Blending (type: integer):

Specifies one of the blending types (0~13) found in the blending type list at the beginning of this chapter.

Can receive input from other nodes in the network or can be specified to by selecting a blending mode from the Blending pull-down menu in the Edit Panel for the node.

For general use specifying the blending mode by using the Blending pull-down menu in the Edit Panel will probably be the most desirable method of use. However, for larger networks where control of the blending modes of many texture nodes simultaneously is needed or when conditional blending is desired then using a node connection to control this value may be advantageous.





## Opacity (type: scalar):

Opacity is the amount that the selected blending mode is applied. You can think of opacity as the strength of the blend effect between the BG Color and the FG Color.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

## U Tiles (type: integer):

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

Specifies the number of tiles in the U dimension.

## V Tiles (type: integer):

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

Specifies the number of tiles in the V dimension.

## U Offset (type: scalar):

This specifies the offset value of the U tiles. This is a little bit like Position but only for the U value of the UV tiles.

Can receive patterns and numerical inputs from other nodes in the network.

## V Offset (type: scalar):

This specifies the offset value of the V tiles. This is a little bit like Position but only for the V value of the UV tiles.

Can receive patterns and numerical inputs from other nodes in the network.

## Bump (type: vector):

Specifies a vector direction for modifying the surface normal. This is surface normal direction information and affects the way the surface is shaded. Care should be taken when connecting to this input. Connecting dissimilar types or non directional vectors may cause the surface to shade wrongly.

Can receive input from other nodes in the network only.

## Bump Amplitude (type: scalar):

Specifies the bump height or "amplitude" of the Bump directional vectors.

Using Alpha or luminance (connecting Color) values from other textures in the network along with their respective bumps can have the effect of blending the bump textures together.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Since Bump Amplitude is a shading feature and not a displacement coordinate system value it is specified as a percentage. Specifically as a percentage of the texture value.

## Scale (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Scale receives its X, Y, Z, scale values either by entering them into the Scale tab in the Edit Panel or by connecting an input to Scale on the node in the Workspace.

Scale is the scale of the texture pattern that will be applied to your object surface.

If the World Coordinates checkbox is checked in the edit panel then Scale defines the scale of the texture in relation to the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, and Z respectively.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures scale is derived from the selected Reference Object keyframed Scale values.

Connections made in the workspace area will override any values entered by the user into the edit panel.

### Position (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Position receives its X, Y, Z, coordinate information either by entering it into the Position tab in the Edit Panel or by connecting an input to the Position connection on the node in the Workspace Area.

Position defines the center coordinate of the texture. By default these are offset values of the objects coordinates also sometimes referred to as the objects pivot point.

If the World Coordinates checkbox is checked in the Edit Panel then Position defines the center of the texture as offset values from the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, Z format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures position is derived from the selected Reference Objects keyframed Position values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.





## Rotation (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Rotation receives H (Heading), P (Pitch), and B (Bank) rotational information either by entering degrees into the Rotation tab in the Edit Panel or as radians by connecting an input to the Rotation connection on the node in the Workspace Area.

Rotation defines the axial rotation around the textures center point. By default these are offset values of the objects a rotational settings.

If the World Coordinates checkbox is checked in the Edit Panel then Rotation values define the texture rotation as offset values from the world coordinates of your scene which should be fixed. You can rely on your world coordinate rotation values to always be 0,0,0, in H, P, B format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures rotation is derived from the selected Reference Objects keyframed Rotation values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Outputs

### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

### Alpha (type: scalar):

Outputs the embedded single channel alpha (grey) information evaluated on a per spot basis.

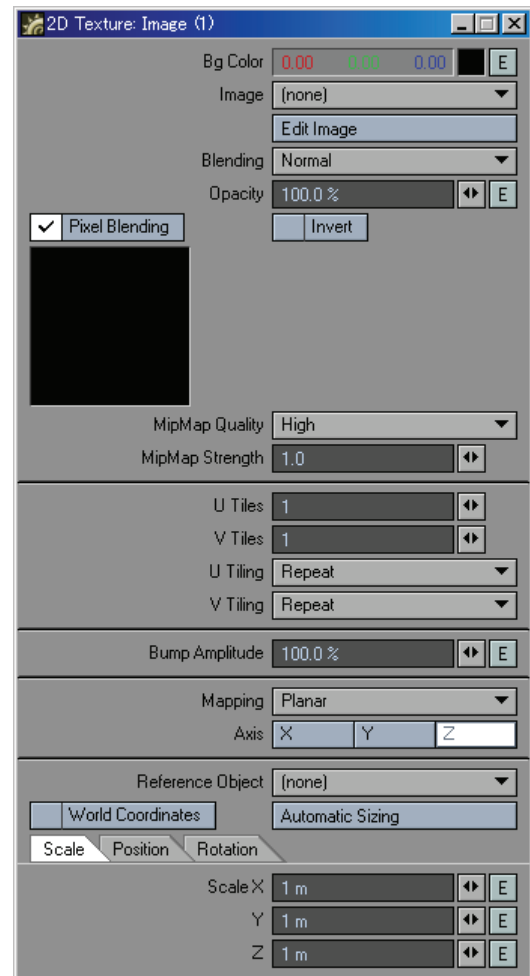
### Luma (type: scalar):

Outputs CCIR 601 luminance values derived from the RGB values contained in the image being used.

### Bump (type: vector):

Outputs component vector direction information the length of which are the product of the bump amplitude value.

## Edit Panel



Here the Mapping projection type and Axis are not offered as node connections in the Workspace area. Likewise Reference Object, World Coordinates and Automatic Sizing are features exclusive to this nodes Edit Panel. Please see the explanation for these items in the chapter that covers texturing with the Surface Editor.

Also please see the related sections elsewhere in this manual for descriptions of Pixel Blending, MipMap Quality, and MipMap Strength.

The explanation of the Image Map Layer Type in the Texture Editor found elsewhere in this manual differ concerning tiling methods, only in that the labels describing the menus are different.

Width Tile in the Texture Editor is U Tiling in the Node Editors' Image node.

Height Tile in the Texture Editor is V Tiling in the Node Editors' Image node.

The behavior described for Reset, Repeat, Mirror and Edge are the same as described for the Texture Editor.



**NOTE:** The image node inverts the alpha, or any value connected to the opacity input of the node, and does not negate the colors in the image like the texture editor.



### Usage Example

This example demonstrates the usefulness of using procedurals with image maps as well exemplifies the process of simple image mapping.



Image Map With Procedural Textures Added.



Image Maps Only Without Procedural Textures

In the top example, procedural textures were used to create a more photo realistic aged look to a simple wood and or nature image map.



Several image map examples are available on the content CD and you are encouraged to take a look at them for a hands on understanding of how they work and what is going on under the hood.

This example uses three Image nodes. The first one is the color map and uses the Cubic projection Mapping method as do all of the Image nodes in this network. Its Color output is being used as the foreground color of the fBm Noise node so that the noise procedural texture blends from the noise pattern color to the wood pattern color. Additionally it is used for the foreground color of the Mixer node. This dual use is important for getting the two textures to mask and blend properly.

A secondary level of blending is achieved by using a gradient feathering technique on an image of the ornate areas alone. This has the effect of mixing the foreground and background layers of the Mixer node in such a way that the dirt and aged effects produced by the fBm Noise (1) node remain close in tight to the ornate areas of the general surface.

This secondary level of blending is itself blended and convoluted by another instance of fBm Noise but at a much larger scale than the one used to produce the dirt color. The Alpha output from fBm Noise (2) is being used both for opacity and to control the U and V offsets which together produce kind of a worm or termite eaten look.

To see the results of this secondary blending select the Surface destination node, press [Shift]+[d], and connect the Luma output from Image (3) to the destination "Surface" Color input. Once completed you'll be able to see the values being used for the Opacity input of the Mixer node. Where the Luma values are high (white) the foreground color will take precedence and where the Luma values are low (black) the background areas will be more predominant.

Lastly, a special grayscale image of the wood-and-or nature image was produced by using Genetica 2.5 Pro, and is being used here as a bump map in order to add shading highlights that produce the surface imperfection or roughness effect of actual wood grain, crevices between the planks and the raised ornature.

To view the surface and object displayed in this manual load the object named "Image\_Cube.lwo" from the Manual Examples\2DTextures folder and open the Node Editor for the surface. Alternatively you may want to apply the "ImageMap Example" preset from the [F8] Surface Preset shelf to an object of your own creation.



## NormalMap

### Purpose

To make use of loaded Normal Map images within a network or as freestanding connections directly connected to the destination node.

Also see the Transform and Transform2 nodes.

### Description

The normal map node allows you to load and utilize normal maps created from any of the various software packages that produce them such as ZBrush, Genetica Pro, or Marvin Landis's free NormalMapCreate plug in for LightWave3D.

With descriptions of normal mapping technology being commonplace we will highlight just a few of the main points you may want to consider when applying normal maps in LightWave3D. Namely what normals are in general, what happens when you apply a normal map, and a few of the differences between normal mapping and bump mapping.

It is easy to understand what normals are if we consider polygon normal or "geometric normals". Polygon normals are simply just the direction in which a given polygon is facing - its facing or "normal" direction. Smoothed normals also consider smoothing interpolation. What's happening when you apply a normal map is the replacement of that information with a set of directions contained in a map. The recorded directions are color encoded (RGB = the directions XYZ). This is different than standard bump mapping in that standard bump mapping does not replace the normal direction but rather modifies the existing directions. This is an important distinction because you're replacing the direction the smooth shaded faces are facing when you apply a normal map and the necessary cautions need to be taken in order to produce correct results.

## Inputs

### U Tiles (type: integer):

Specifies the number of tiles per scale value in the U direction.

Can receive patterns and numerical inputs from other nodes in the network. Additionally, controls offered in the Edit Panel for this node may be used to specify this value.

### V Tiles (type: integer):

Specifies the number of tiles per scale value in the V direction.

Can receive patterns and numerical inputs from other nodes in the network. Additionally, controls offered in the Edit Panel for this node may be used to specify this value.

### U Offset (type: scalar):

This specifies the offset value of the U tiles. This is a little bit like Position but only for the U value of the UV tiles.

Can receive patterns and numerical inputs from other nodes in the network.

### V Offset (type: scalar):

This specifies the offset value of the V tiles. This is a little bit like Position but only for the V value of the UV tiles.

Can receive patterns and numerical inputs from other nodes in the network.

### Amplitude (type: scalar):

Specifies the normal height or "amplitude" of the Normal directional vectors.

Using Alpha or luminance (connecting Color) values from other textures in the network along with their respective bumps can have the effect of blending the bump textures together.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Since Bump Amplitude is a shading feature and not a displacement coordinate system value it is specified as a percentage. Specifically as a percentage of the texture value.

### Scale (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Scale receives its X, Y, Z, scale values either by entering them into the Scale tab in the Edit Panel or by connecting an input to Scale on the node in the Workspace.

Scale is the scale of the texture pattern that will be applied to your object surface.



If the World Coordinates checkbox is checked in the edit panel then Scale defines the scale of the texture in relation to the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, and Z respectively.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures scale is derived from the selected Reference Object keyframed Scale values.

Connections made in the workspace area will override any values entered by the user into the edit panel.

Position (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Position receives its X. Y. Z. coordinate information either by entering it into the Position tab in the Edit Panel or by connecting an input to the Position connection on the node in the Workspace Area.

Position defines the center coordinate of the texture. By default these are offset values of the objects coordinates also sometimes referred to as the objects pivot point.

If the World Coordinates checkbox is checked in the Edit Panel then Position defines the center of the texture as offset values from the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, Z format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures position is derived from the selected Reference Objects keyframed Position values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Rotation (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Rotation receives H (Heading), P (Pitch), and B (Bank) rotational information either by entering degrees into the Rotation tab in the Edit Panel or as radians by connecting an input to the Rotation connection on the node in the Workspace Area.

Rotation defines the axial rotation around the textures center point. By default these are offset values of the objects a rotational settings.

If the World Coordinates checkbox is checked in the Edit Panel then Rotation values define the texture rotation as offset values from the world coordinates of your scene which should be fixed. You can rely on your world coordinate rotation values to always be 0,0,0, in H, P, B format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures rotation is derived from the selected Reference Objects keyframed Rotation values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

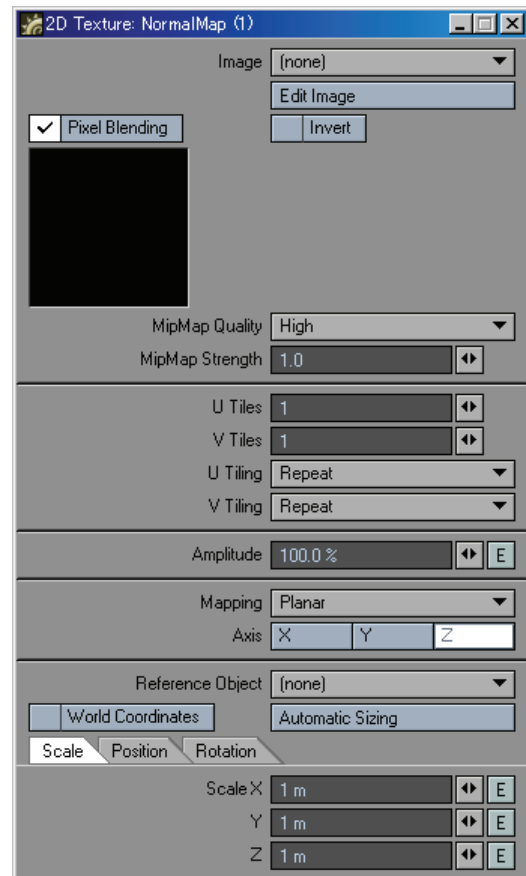
## Outputs

Normal (type: vector):

Outputs component vector direction information the length of which are the product of the bump amplitude value.

If used as an input to another normal type connection the destinations normal will be completely replaced with this information.

## Edit Panel



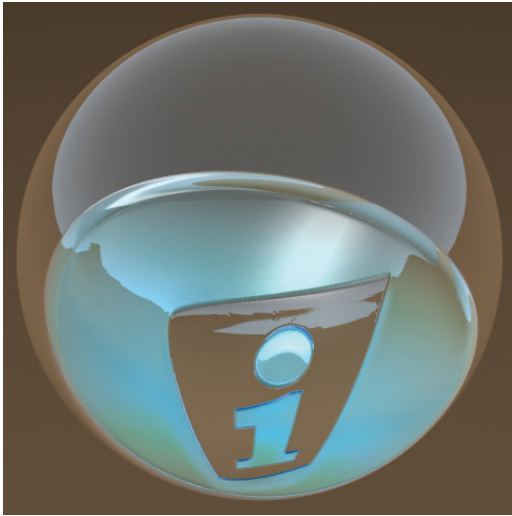
Here the Mapping projection type and Axis are not offered as node connections in the Workspace area. Likewise Reference Object, World Coordinates and Automatic Sizing are features exclusive to this nodes Edit Panel. Please see the explanation for these items in the chapter that covers texturing with the Surface Editor.

Also please see the relative sections elsewhere in this manual for descriptions of Pixel Blending, MipMap Quality, and MipMap Strength.

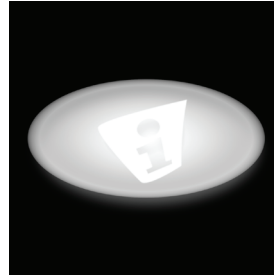


## Usage Example

Here is a simple example using the NormalMap node to apply a normal map onto a sphere. The result is apparent detail without the need for added geometry beyond the polygons of the sphere that already exist.



The color map is being used to affect the surface color attributes. Its interpolated luminance values define the Transparency and the amount of Color Filtering.



The embedded Alpha map here removed, is defining the amount of luminosity being applied to the surface.

The images, objects, and presets used in this example may be found in the LightWave3D content folder installed with your application.



This example uses an image with embedded alpha and a normal map image that were created in Genetica 2.5 Pro.



The normal map was loaded into the NormalMap node and its Normal output connected directly to the Normal input of the Surface destination node.







## Parquet2D

### Description

Parquetry is the inlay of wood, often of different colors, that is worked into a geometric pattern or mosaic and is used especially for floors. The Parquet2D procedural texture offers the ability to create these geometric patterns. While a very common use would be for floor designs, the Parquet2D texture like other procedural textures, affords a great deal of creativity limited only by the imagination of the user. Parquet2D has axial projection methods available such as Spherical, Cubic, Planar, Front, Cylindrical and UV.

### Inputs

#### Bg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the background color layer.

#### Fg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the foreground color layer.

#### Blending (type: integer):

Specifies one of the blending types (0~13) found in the blending type list at the beginning of this chapter.

Can receive input from other nodes in the network or can be specified to by selecting a blending mode from the Blending pull-down menu in the Edit Panel for the node.

For general use specifying the blending mode by using the Blending pull-down menu in the Edit Panel will probably be the most desirable method of use. However, for larger networks where control of the blending modes of many texture nodes simultaneously is needed or when conditional blending is desired then using a node connection to control this value may be advantageous.

#### Opacity (type: scalar):

Opacity is the amount that the selected blending mode is applied. You can think of opacity as the strength of the blend effect between the BG Color and the FG Color.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Tiles (type: integer):

This specifies the number of parallel planks or sections that each parquet tile is comprised of.

Values below one will be evaluated as one.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### U Tiles (type: integer):

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

Specifies the number of tiles in the U dimension.

#### V Tiles (type: integer):

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

Specifies the number of tiles in the V dimension.

#### Mortar Width (type: scalar):

Specifies the width of the mortar (or gaps) between the parquet faces. Increasing this value will also reduce the size of the face area in such a way that the pattern will repeat correctly the number of U and V tiles within the defined scale of the texture.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Mortar Sharp (type: scalar):

Defines how steep the falloff angle is between the parquet faces and the bottom trough of the mortar (or gap) area.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### U Offset (type: scalar):

This specifies the offset value of the U tiles. This is a little bit like Position but only for the U value of the UV tiles.

Can receive patterns and numerical inputs from other nodes in the network.

#### V Offset (type: scalar):

This specifies the offset value of the V tiles. This is a little bit like Position but only for the V value of the UV tiles.

Can receive patterns and numerical inputs from other nodes in the network.





## Function (type: function):

Connecting a function node to this input will transform the texture value of this texture based on the graph defined by the function thereby extending the range of this nodes producible patterns. For example bevels and so forth can be added. See the manual section on Function nodes for a description of how texture values can be transformed.

Can receive inputs from other function nodes in the network only.

## Bump (type: vector):

Specifies a vector direction for modifying the surface normal. This is surface normal direction information and affects the way the surface is shaded. Care should be taken when connecting to this input. Connecting dissimilar types or non directional vectors may cause the surface to shade wrongly.

Can receive input from other nodes in the network only.

## Bump Amplitude (type: scalar):

Specifies the bump height or "amplitude" of the Bump directional vectors.

Using Alpha or luminance (connecting Color) values from other textures in the network along with their respective bumps can have the effect of blending the bump textures together.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Since Bump Amplitude is a shading feature and not a displacement coordinate system value it is specified as a percentage. Specifically as a percentage of the texture value.

## Scale (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Scale receives its X, Y, Z, scale values either by entering them into the Scale tab in the Edit Panel or by connecting an input to Scale on the node in the Workspace.

Scale is the scale of the texture pattern that will be applied to your object surface.

If the World Coordinates checkbox is checked in the edit panel then Scale defines the scale of the texture in relation to the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, and Z respectively.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures scale is derived from the selected Reference Object keyframed Scale values.

Connections made in the workspace area will override any values entered by the user into the edit panel.

## Position (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Position receives its X, Y, Z. coordinate information either

by entering it into the Position tab in the Edit Panel or by connecting an input to the Position connection on the node in the Workspace Area.

Position defines the center coordinate of the texture. By default these are offset values of the objects coordinates also sometimes referred to as the objects pivot point.

If the World Coordinates checkbox is checked in the Edit Panel then Position defines the center of the texture as offset values from the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, Z format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures position is derived from the selected Reference Objects keyframed Position values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Rotation (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Rotation receives H (Heading), P (Pitch), and B (Bank) rotational information either by entering degrees into the Rotation tab in the Edit Panel or as radians by connecting an input to the Rotation connection on the node in the Workspace Area.

Rotation defines the axial rotation around the textures center point. By default these are offset values of the objects a rotational settings.

If the World Coordinates checkbox is checked in the Edit Panel then Rotation values define the texture rotation as offset values from the world coordinates of your scene which should be fixed. You can rely on your world coordinate rotation values to always be 0,0,0, in H, P, B format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures rotation is derived from the selected Reference Objects keyframed Rotation values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Outputs

### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

### Alpha (type: scalar):

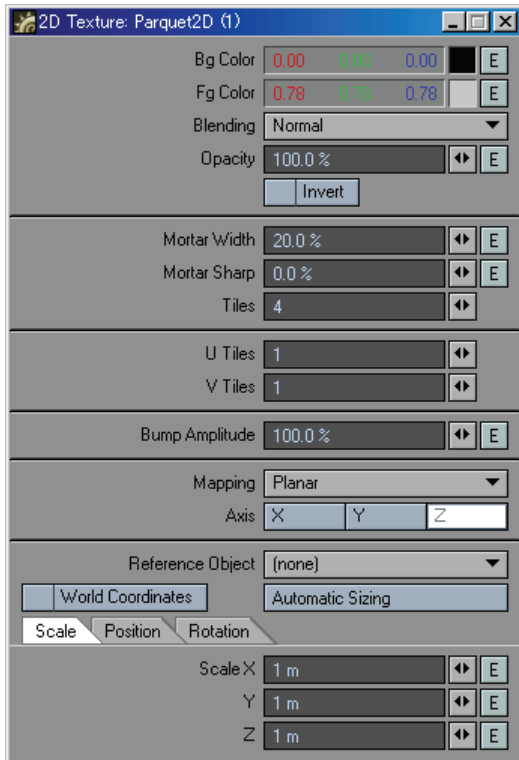
Outputs single channel alpha (grey) information evaluated on a per spot basis.

### Bump (type: vector):

Outputs component vector direction information the length of which are the product of the bump amplitude value.



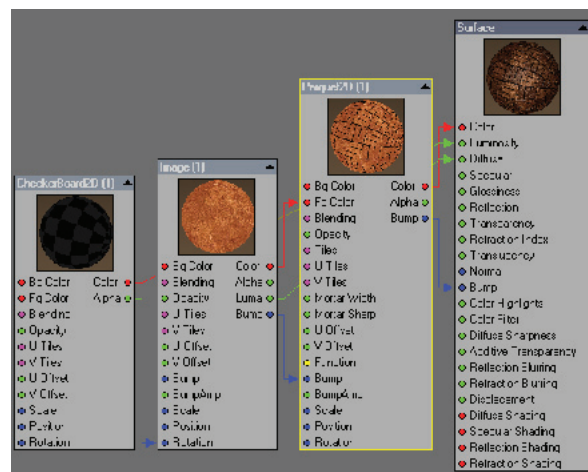
## Edit Panel



Here the Mapping projection type and Axis are not offered as node connections in the Workspace area. Likewise Reference Object, World Coordinates and Automatic Sizing are features exclusive to this nodes Edit Panel. Please see the explanation for these items in the chapter that covers texturing with the Surface Editor.

## Usage Example

This is a fairly straightforward use of the Parquet2D procedural texture node.



Of interest in this example besides of course the generic use of spherically mapped parquet tiles, is that here again the checkerboard is being used both to rotate sections of an image (from the Image node) to fit the same tile sizes created by the Parquet node, and also to slightly alter the luminance values of the surface giving the appearance of two different materials from a single image file.

The luminance values of the image map are connected to the Diffuse input of the Surface destination node in order to accentuate the basic character of the leather pattern image being used.

The bump map extrapolated from the image file being used is mixed with the procedurally generated bump map produced by the Parquet2D node. This is achieved simply by connecting the Bump output from the Image node to the Bump input of the Parquet2D node, which then finally is connected to the Surface destination Bump input. Adding bump maps together in this way can be very handy when several or more bumps downstream of one another need to be mixed to achieve a particular result.



## Planks2D

### Description

The Planks2D procedural texture creates the gaps and faces of flush or staggered planks. Like Parquet2D, Bricks2D, Bricks3D and other procedurals of this type a separate texture either procedural or image based, can be used where face and grout (mortar, gaps, etc.) materials are desired. Planks2D has axial projection methods available such as spherical, cubic, planar, front, cylindrical and UV.

### Inputs

#### Bg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the background color layer. In this case the gaps between the faces.

#### Fg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the foreground color layer. In this case the plank faces.

#### Blending (type: integer):

Specifies one of the blending types (0~13) found in the blending type list at the beginning of this chapter.

Can receive input from other nodes in the network or can be specified to by selecting a blending mode from the Blending pull-down menu in the Edit Panel for the node.

For general use specifying the blending mode by using the Blending pull-down menu in the Edit Panel will probably be the most desirable method of use. However, for larger networks where control of the blending modes of many texture nodes simultaneously is needed or when conditional blending is desired then using a node connection to control this value may be advantageous.

#### Opacity (type: scalar):

Opacity is the amount that the selected blending mode is applied. You can think of opacity as the strength of the blend effect between the BG Color and the FG Color.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### U Tiles (type: integer):

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

Specifies the number of tiles in the U dimension.

#### V Tiles (type: integer):

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

Specifies the number of tiles in the V dimension.

#### Mortar Width (type: scalar):

Specifies the width of the mortar (or gaps) between the plank faces. Increasing this value will also reduce the size of the face area in such a way that the pattern will repeat correctly the number of U and V tiles within the defined Scale of the texture.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Mortar Sharp (type: scalar):

Defines how steep the falloff angle is between the plank faces and the bottom trough of the mortar (or gap) area.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Stagger (type: scalar):

Stagger specifies the offset of the U directional gap (or "mortar") areas. A value of 0% will cause all of the planks to line up flush on both ends forming a straight row of planks.

Can receive patterns and numerical inputs from other nodes in the network. Additionally user input values may be entered in the Edit Panel for the node.

#### Plank Length (type: scalar):

Specifies the length of the planks as a percentage to the overall size of the texture.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.



## U Offset (type: scalar):

This specifies the offset value of the U tiles. This is a little bit like Position but only for the U value of the UV tiles.

Can receive patterns and numerical inputs from other nodes in the network.

## V Offset (type: scalar):

This specifies the offset value of the V tiles. This is a little bit like Position but only for the V value of the UV tiles.

Can receive patterns and numerical inputs from other nodes in the network.

## Function (type: function):

Connecting a function node to this input will transform the texture value of this texture based on the graph defined by the function thereby extending the range of this nodes producible patterns. For example bevels and rounded edges can be added to the pattern information. See the manual section on Function nodes for a description of how texture values can be transformed.

Can receive inputs from other function nodes in the network only.

## Bump (type: vector):

Specifies a vector direction for modifying the surface normal. This is surface normal direction information and affects the way the surface is shaded. Care should be taken when connecting to this input. Connecting dissimilar types or non directional vectors may cause the surface to shade wrongly.

Can receive input from other nodes in the network only.

## Bump Amplitude (type: scalar):

Specifies the bump height or "amplitude" of the Bump directional vectors.

Using Alpha or luminance (connecting Color) values from other textures in the network along with their respective bumps can have the effect of blending the bump textures together.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Since Bump Amplitude is a shading feature and not a displacement coordinate system value it is specified as a percentage. Specifically as a percentage of the texture value.

## Scale (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Scale receives its X, Y, Z, scale values either by entering them into the Scale tab in the Edit Panel or by connecting an input to Scale on the node in the Workspace.

Scale is the scale of the texture pattern that will be applied to your object surface.

If the World Coordinates checkbox is checked in the edit panel then Scale defines the scale of the texture in relation to the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, and Z respectively.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures scale is derived from the selected Reference Object keyframed Scale values.

Connections made in the workspace area will override any values entered by the user into the edit panel.

## Position (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Position receives its X, Y, Z, coordinate information either by entering it into the Position tab in the Edit Panel or by connecting an input to the Position connection on the node in the Workspace Area.

Position defines the center coordinate of the texture. By default these are offset values of the objects coordinates also sometimes referred to as the objects pivot point.

If the World Coordinates checkbox is checked in the Edit Panel then Position defines the center of the texture as offset values from the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, Z format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures position is derived from the selected Reference Objects keyframed Position values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Rotation (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Rotation receives H (Heading), P (Pitch), and B (Bank) rotational information either by entering degrees into the Rotation tab in the Edit Panel or as radians by connecting an input to the Rotation connection on the node in the Workspace Area.

Rotation defines the axial rotation around the textures center point. By default these are offset values of the objects a rotational settings.

If the World Coordinates checkbox is checked in the Edit Panel then Rotation values define the texture rotation as offset values from the world coordinates of your scene which should be fixed. You can rely on your world coordinate rotation values to always be 0,0,0, in H, P, B format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures rotation is derived from the selected Reference Objects keyframed Rotation values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.



## Outputs

### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

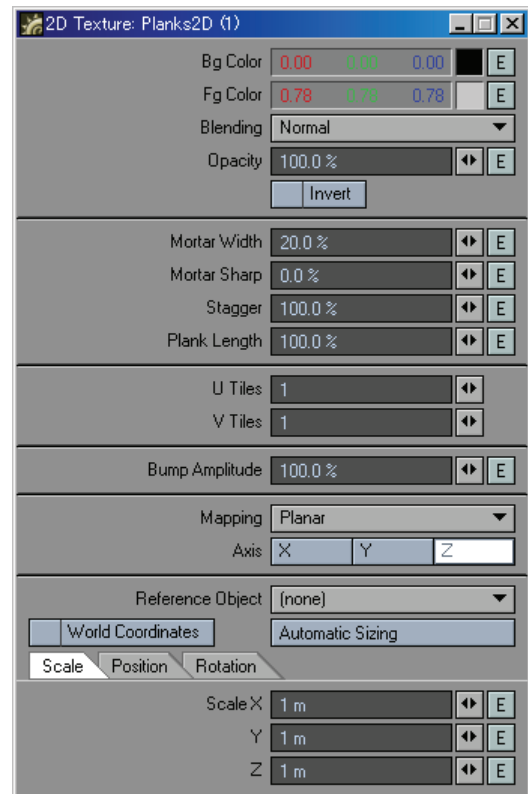
### Alpha (type: scalar):

Outputs single channel alpha (grey) information evaluated on a per spot basis.

### Bump (type: vector):

Outputs component vector direction information the length of which are the product of the bump amplitude value.

## Edit Panel



Here the Mapping projection type and Axis are not offered as node connections in the Workspace area. Likewise Reference Object, World Coordinates and Automatic Sizing are features exclusive to this nodes Edit Panel. Please see the explanation for these items in the chapter that covers texturing with the Surface Editor.



## Usage Example

Using the Planks2D procedural node over an image of seamlessly repeating, infinite length planks, to add cross cut lengths and shore up any length-gap imperfections:

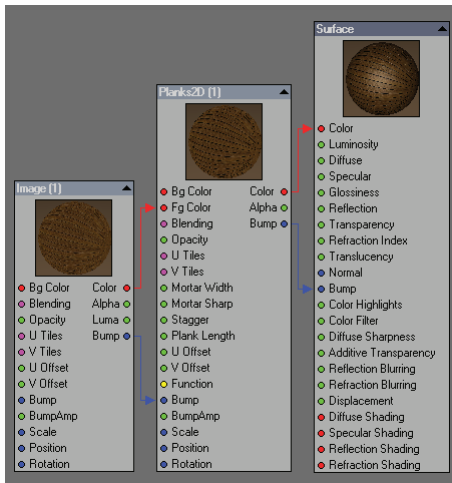


This example then offers an extra utilitarian aspect. One of cleaning up and enhancing a photographic texture and all the imperfections associated with real world lighting, lenses, and so forth.

On the left is the base image that was used. At center, the planks procedural color pattern that was used to enhance the photograph. And on the right, a relief rendering of the combined bump maps from both the wood planks photo and the Planks2D procedural:



Together they form more believable wooden planks than either of the component's individually.



Of course used creatively this procedural can create bizarre and wild patterns but with the artistic touch aside for a moment let's examine just one of many of its more conventional applications.

Here we have taken an existing photograph of wooden planks and basically stenciled both color and bump channels from the Planks2D procedural on top of it.

Normally this would not need to be a photograph of *planks* specifically. It could easily be a photograph or even a procedural of any wood (or alike material) texture where scaling the two textures in relation would offer the keys to believability.





## Turbulence2D

### Description

A multipurpose noise texture with axial projection methods such as spherical cubic planar front cylindrical and UV.

### Inputs

#### Bg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the background color layer.

#### Fg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the foreground color layer.

#### Blending (type: integer):

Specifies one of the blending types (0~13) found in the blending type list at the beginning of this chapter.

Can receive input from other nodes in the network or can be specified to by selecting a blending mode from the Blending pull-down menu in the Edit Panel for the node.

For general use specifying the blending mode by using the Blending pull-down menu in the Edit Panel will probably be the most desirable method of use. However, for larger networks where control of the blending modes of many texture nodes simultaneously is needed or when conditional blending is desired then using a node connection to control this value may be advantageous.

#### Opacity (type: scalar):

Opacity is the amount that the selected blending mode is applied. You can think of opacity as the strength of the blend effect between the BG Color and the FG Color.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### U Tiles (type: integer):

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

Specifies the number of tiles in the U dimension.

#### V Tiles (type: integer):

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

Specifies the number of tiles in the V dimension.

#### Small Scale (type: scalar):

Like Lacunarity, Small Scale is a counterpart to the fractal dimension that describes the texture of a fractal. It has to do with the size distribution of the gaps between detail levels. The Small Scale value sets the amount of change in scale between successive levels of detail.

If you're seeing repeating patterns in the texture it might be a good idea to alter the Small Scale value for the texture.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify values for this attribute using the controls available in the Edit Panel for this node.

#### Contrast (type: scalar):

Sets the contrast of the texture value gradient. Higher values produce less gradient information between texture value extremes and therefore less noise is produced. The lower values spread the gradient more evenly between the extremes of the texture values and thus allow for more overall noise in the final output from this node.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify the values for this attribute using the controls available in the Edit Panel for this node.

#### Frequencies (type: scalar):

"Frequencies" as used in this node is really just another name for octaves as defined in some of the descriptions of fractal type textures within this document.

You can think of "Frequencies" as the number of levels of detail mentioned in the discussion just above for Small Scale. At one "Frequency", only the basic pattern is used. At 2 "Frequencies" given a Small Scale value of 2, a one half scale pattern is overlaid on top of the basic pattern. And at three Frequencies, the overlays are one quarter scale over the one half scale over the basic and so on.

If Small Scale were 3 in this example, it would mean that each new iteration would be one third the scale of the previous layer.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify the values for this attribute using the controls available in the Edit Panel for this node.



## U Offset (type: scalar):

This specifies the offset value of the U tiles. This is a little bit like Position but only for the U value of the UV tiles.

Can receive patterns and numerical inputs from other nodes in the network.

## V Offset (type: scalar):

This specifies the offset value of the V tiles. This is a little bit like Position but only for the V value of the UV tiles.

Can receive patterns and numerical inputs from other nodes in the network.

## Function (type: function):

Connecting a function node to this input will transform the texture value of this texture based on the graph defined by the function thereby extending the range of this nodes producible patterns. See the manual section on Function nodes for a description of how texture values can be transformed.

Can receive inputs from other function nodes in the network only.

## Bump (type: vector):

Specifies a vector direction for modifying the surface normal. This is surface normal direction information and affects the way the surface is shaded. Care should be taken when connecting to this input. Connecting dissimilar types or non directional vectors may cause the surface to shade wrongly.

Can receive input from other nodes in the network only.

## Bump Amplitude (type: scalar):

Specifies the bump height or "amplitude" of the Bump directional vectors.

Using Alpha or luminance (connecting Color) values from other textures in the network along with their respective bumps can have the effect of blending the bump textures together.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Since Bump Amplitude is a shading feature and not a displacement coordinate system value it is specified as a percentage. Specifically as a percentage of the texture value.

## Scale (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Scale receives its X, Y, Z, scale values either by entering them into the Scale tab in the Edit Panel or by connecting an input to Scale on the node in the Workspace.

Scale is the scale of the texture pattern that will be applied to your object surface.

If the World Coordinates checkbox is checked in the edit panel then Scale defines the scale of the texture in relation to the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, and Z respectively.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures scale is derived from the selected Reference Object keyframed Scale values.

Connections made in the workspace area will override any values entered by the user into the edit panel.

### Position (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Position receives its X, Y, Z, coordinate information either by entering it into the Position tab in the Edit Panel or by connecting an input to the Position connection on the node in the Workspace Area.

Position defines the center coordinate of the texture. By default these are offset values of the objects coordinates also sometimes referred to as the objects pivot point.

If the World Coordinates checkbox is checked in the Edit Panel then Position defines the center of the texture as offset values from the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, Z format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures position is derived from the selected Reference Objects keyframed Position values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Rotation (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Rotation receives H (Heading), P (Pitch), and B (Bank) rotational information either by entering degrees into the Rotation tab in the Edit Panel or as radians by connecting an input to the Rotation connection on the node in the Workspace Area.



Rotation defines the axial rotation around the textures center point. By default these are offset values of the objects a rotational settings.

If the World Coordinates checkbox is checked in the Edit Panel then Rotation values define the texture rotation as offset values from the world coordinates of your scene which should be fixed. You can rely on your world coordinate rotation values to always be 0,0,0, in H, P, B format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures rotation is derived from the selected Reference Objects keyframed Rotation values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Outputs

### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

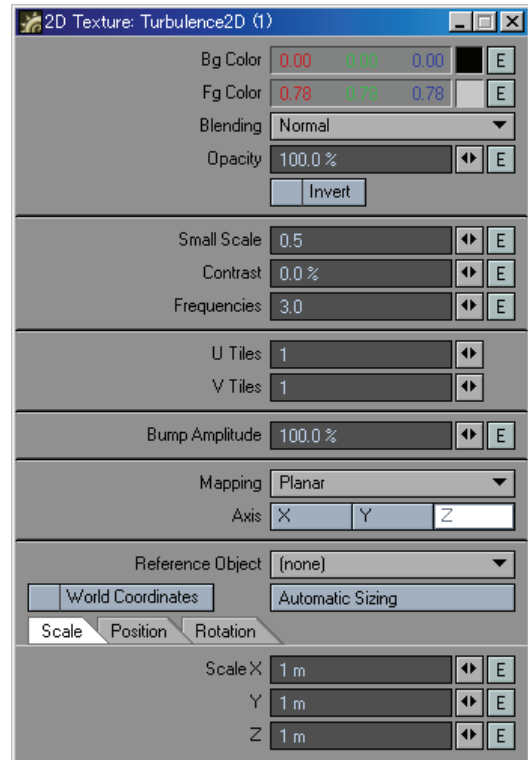
### Alpha (type: scalar):

Outputs single channel alpha (grey) information evaluated on a per spot basis.

### Bump (type: vector):

Outputs component vector direction information the length of which are the product of the bump amplitude value.

## Edit Panel

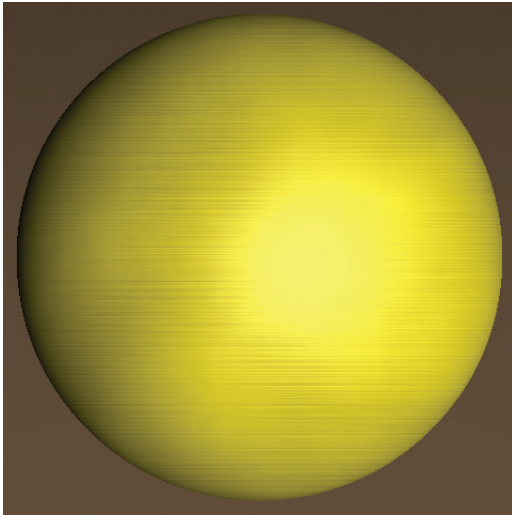


Here the Mapping projection type and Axis are not offered as node connections in the Workspace area. Likewise Reference Object, World Coordinates and Automatic Sizing are features exclusive to this nodes Edit Panel. Please see the explanation for these items in the chapter that covers texturing with the Surface Editor.



## Usage Example

Using turbulence for scratches and dirt.



The Turbulence2D texture and its 3D counterpart (Turbulence), are very commonly used in the computer graphics industry across a wide variety of 3D applications. Among some of the more popular uses for turbulence are its application in “breaking up” a textures austere appositional uniformity.

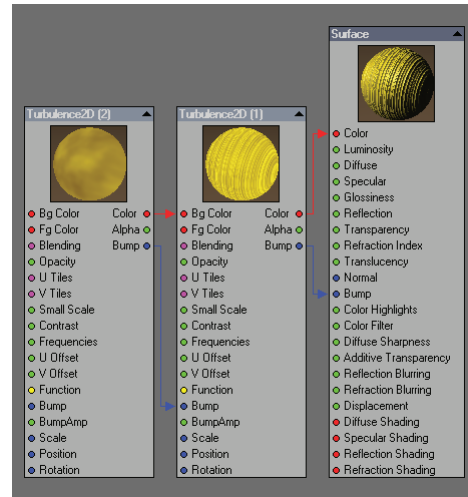
Because of its 2D factor Turbulence2D becomes a natural choice for applying to UV maps for the reasons we just discussed. Again, as with any texture its uses are limited only by your imagination.

In this example we are using Turbulence2D in two very specific and unique ways. The first is for linear scratches as one might to observe on a playground slide or factory shoot.

This is achieved by either scaling one of the two axes not being used for the axial projection or by creating a large ratio between the number of U and V tiles. For example in the case of using the textures Scale values to achieve this result you would want to scale either an X or Z very small in relationship to the other if your projection axis was set to Y. Likewise you would make either Y. or Z. small if you're axis was set to X, etc.

This example uses a high ratio between the number of U and V tiles to create the apparent scratches you see in the rendered example. Often this technique is applied only to the Surface nodes' bump input in order to affect the specular hits produced by camera and light positions when animated.

The second way that turbulence is being used here can be discovered by examining the “Turbulence2D (2)” node. Basically to flatten out and exaggerate different areas of the linear scratches that are applied to Color and Bump of the surface, thus reducing its uniformity. The larger the surface area that the scratches are applied to the larger you'll want to make the scale of the Turbulence2D (2) node for the affect to be believable in a typical application.



Where the U Tiles are set to 1 and the V Tiles are set to 400, Turbulence2D (1) is producing the linear scratch effect you see in the example.

Turbulence2D (2) set at 3 tiles for both U Tiles and V Tiles, giving the surface its somewhat patchy uneven look.



---

## 3D Textures

---

A 3D texture is computer generated via procedural algorithms with a set of parameters. The generated pattern follows logically inside the procedural volume and wherever polygonal surfaces exist in relation to that volume the procedural pattern information is applied. 3D textures can exist in world space coordinates or tied to an object's center point when world coordinates is unchecked in the node's Edit Panel. 3D textures avoid many of the difficulties attributed to the mapping of a bitmap around the surface of an irregular object. These textures have no edges and provide a continuous-looking appearance.

---

## Bricks

---

### Description

Bricks is a brick-like 3D texture useful in producing many patterns and effects. Modified, many elaborate variations diverging from conventionally recognizable brick patterns can be achieved.

### Inputs

#### Bg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the background color layer.

#### Fg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the foreground color layer.

#### Blending (type: integer):

Specifies one of the blending types (0~13) found in the blending type list at the beginning of this chapter.

Can receive input from other nodes in the network or can be specified to by selecting a blending mode from the Blending pull-down menu in the Edit Panel for the node.

For general use specifying the blending mode by using the Blending pull-down menu in the Edit Panel will probably be the most desirable method of use. However, for larger networks where control of the blending modes of many texture nodes simultaneously is needed or when conditional blending is desired then using a node connection to control this value may be advantageous.

#### Opacity (type: scalar):

Opacity is the amount that the selected blending mode is applied. You can think of opacity as the strength of the blend effect between the BG Color and the FG Color.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.



## Thickness (type: scalar):

Is the thickness of the mortar areas of the brick pattern. As the thickness value increases the size of the brick faces decrease within a given texture scale creating the appearance of smaller bricks spaced farther apart.

This value is in ratio to the texture scale. For example a value of 1.0 would produce a surface that was nothing but mortar with no visible brick faces. A value of zero would produce a surface of continuous brick faces with no mortar gaps and only hairline cracks that might be seen in very high precision stonework.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this value using the controls available in the Edit Panel for this node.

## Edge Width (type: scalar):

Is the width of the beveled edges which create a rounded or beveled look to the four sides of the brick face.

This setting in ratio to the size of the brick face and is calculated as an offset along each side of each brick face.

For example a value of 0.5 would produce a pyramid shape that would occupy the entire face area of each and every brick because 0.5 or halfway in from each edge of the brick is the entire brick face.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this value using the controls available in the Edit Panel for this node.

## Function (type: function):

Connecting a function node to this input will transform the texture value of this texture based on the graph defined by the function thereby extending the range of this nodes producible patterns. For example beveled or rounded edges can be added to the pattern information.

See the manual section on Function nodes for a description of how texture values can be transformed.

Can receive inputs from other function nodes in the network only.

## Bump (type: vector):

Specifies a vector direction for modifying the surface normal. This is surface normal direction information and affects the way the surface is shaded. Care should be taken when connecting to this input. Connecting dissimilar types or non directional vectors may cause the surface to shade wrongly.

Can receive input from other nodes in the network only.

## Bump Amplitude (type: scalar):

Specifies the bump height or "amplitude" of the Bump directional vectors.

Using Alpha or luminance (connecting Color) values from other textures in the network along with their respective bumps can have the effect of blending the bump textures together.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Since Bump Amplitude is a shading feature and not a displacement coordinate system value it is specified as a percentage. Specifically as a percentage of the texture value.

## Scale (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Scale receives its X, Y, Z, scale values either by entering them into the Scale tab in the Edit Panel or by connecting an input to Scale on the node in the Workspace.

Scale is the scale of the texture pattern that will be applied to your object surface.

If the World Coordinates checkbox is checked in the edit panel then Scale defines the scale of the texture in relation to the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, and Z respectively.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures scale is derived from the selected Reference Object keyframed Scale values.

Connections made in the workspace area will override any values entered by the user into the edit panel.

## Position (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Position receives its X, Y, Z, coordinate information either by entering it into the Position tab in the Edit Panel or by connecting an input to the Position connection on the node in the Workspace Area.

Position defines the center coordinate of the texture. By default these are offset values of the objects coordinates also sometimes referred to as the objects pivot point.

If the World Coordinates checkbox is checked in the Edit Panel then Position defines the center of the texture as offset values from the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, Z format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures position is derived from the selected Reference Objects keyframed Position values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.





## Rotation (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Rotation receives H (Heading), P (Pitch), and B (Bank) rotational information either by entering degrees into the Rotation tab in the Edit Panel or as radians by connecting an input to the Rotation connection on the node in the Workspace Area.

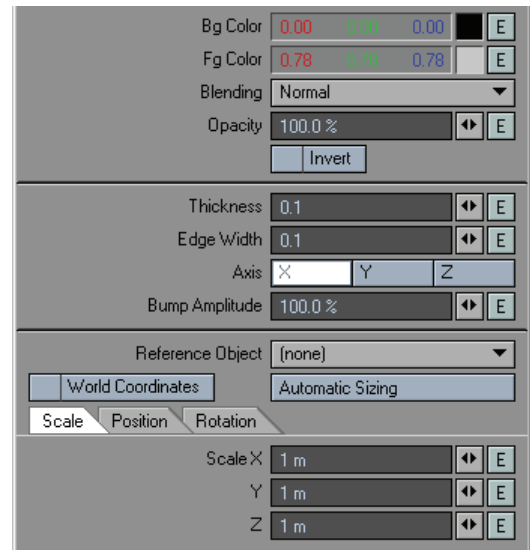
Rotation defines the axial rotation around the textures center point. By default these are offset values of the objects a rotational settings.

If the World Coordinates checkbox is checked in the Edit Panel then Rotation values define the texture rotation as offset values from the world coordinates of your scene which should be fixed. You can rely on your world coordinate rotation values to always be 0,0,0, in H, P, B format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures rotation is derived from the selected Reference Objects keyframed Rotation values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Edit Panel



Axis are not offered as node connections in the Workspace area. Likewise Reference Object, World Coordinates and Automatic Sizing are features exclusive to this nodes Edit Panel. Please see the explanation for these items in the chapter that covers texturing with the Surface Editor.

## Outputs

### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

### Alpha (type: scalar):

Outputs single channel alpha (grey) information evaluated on a per spot basis.

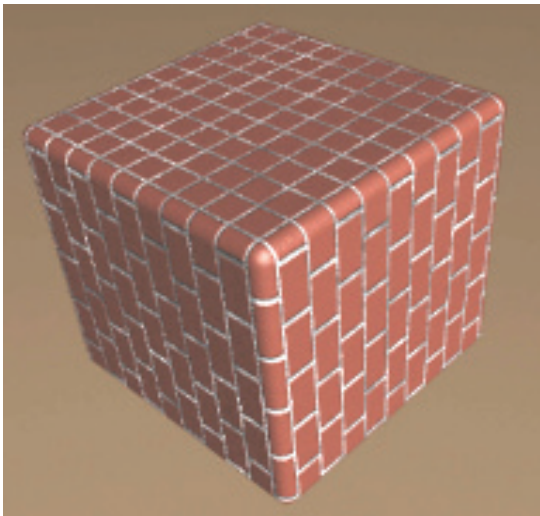
### Bump (type: vector):

Outputs component vector direction information the length of which are the product of the bump amplitude value.



## Usage Example

Using color luminance levels to control the bump amplitude of brick and mortar.



This example demonstrates how easy it is to apply the bump mapping from one texture node only to specific areas of another.

First we add a Brick node and set up the basics - like colors, texture scale, and face and mortar sizes.

Hooking up the Color output of a texture node to the Color input of the destination node while leaving VIPER open makes it very easy and interactive to set up these initial properties.

After we have the basic brick pattern it's a simple task to texture and bump the individual regions - in this case the mortar areas and the brick faces.

We have chose to use the Turbulent Noise texture found in the 3D Textures sub-menu as the source for the color and bump of our mortar areas. After adding the Turbulent Noise node connect its Color output to both the Bg Color and the Fg Color of the Bricks node we just added.

With no outputs other than Color from the Brick node connected to the surface destination this effectively overwrites or "bypasses" the information we set up in the Bricks node.

Using VIPER in the same way we did with Bricks to get interactive feedback while making adjustments, adjust the parameters of the Turbulent Noise until you have a surface that you feel looks something like actual brick mortar. Finally disconnect the Color output of the Turbulent Noise node from the Fg color of the Bricks node.

See the 3D Bricks Example preset in the Manual\_Examples library of the Surface Preset [F8] that was added during the installation process if you're interested in the specific parameter values used in this example.

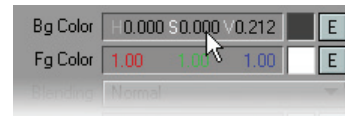
Let's enhance the shading properties by adding a bump map. Connect the Bump output from the Turbulent Noise node to the Brick nodes' Bump input and connect the Bump output from the Brick node to the Bump input of the Surface destination node. Notice how the bump maps across the entire surface evenly over the brick faces and mortar.

We want a very rough bump map for the mortar areas and a very subtle bump map for the brick face areas. Select the Bricks node and press [Ctrl+c] followed by [Ctrl+v] to create a second instance

of our brick pattern with exactly the same settings.

Connect the Color output from the newly pasted Bricks (2) node to the BumpAmp input of the Turbulent Noise node.

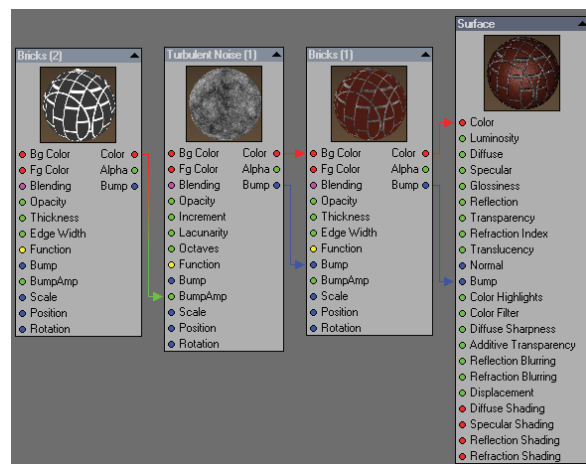
In the Edit Panel of the Bricks (2) node right click inside the color channel value box to alternate to the HSV value display.



Do this for both the Bg Color and the Fg Color and set both (H)ue and (S)aturation levels to 0. When a color output is connected to a scalar input it's luminance values are derived. Working with and modifying only the Value levels in the HSV model will help us visualize the controls we are adjusting.

Essentially we are using these Values to control the amplitude of our Turbulent Noise bump. Since the Bricks (2) node is an identical copy of the one being used to define the surface color decreasing the Value of the Bg Color will produce more subtle or less "amplified" bumps on in the brick face areas. Increasing the Value of the Fg Color will produce stronger rougher bumps in our mortar areas.

Adjust the value for both the background color and the foreground color until you're satisfied with the bump strength in the VIPER preview window.



When finished set the Antialiasing passes in the Camera Properties panel [C], to 5 or more and press [F9] to get an idea of what the final texture will look like when rendered.



**NOTE:** To adjust only the Value component of the HSV model for either the foreground or background colors right click to alternate between the available color models and then left click on the Value you wish to modify and drag your mouse right or left in order to increase or decrease the value respectively.



## Checkerboard

### Description

Procedurally generated checkerboard pattern.

### Inputs

#### Bg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the background color layer.

#### Fg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the foreground color layer.

#### Blending (type: integer):

Specifies one of the blending types (0~13) found in the blending type list at the beginning of this chapter.

Can receive input from other nodes in the network or can be specified to by selecting a blending mode from the Blending pull-down menu in the Edit Panel for the node.

For general use specifying the blending mode by using the Blending pull-down menu in the Edit Panel will probably be the most desirable method of use. However, for larger networks where control of the blending modes of many texture nodes simultaneously is needed or when conditional blending is desired then using a node connection to control this value may be advantageous.

#### Opacity (type: scalar):

Opacity is the amount that the selected blending mode is applied. You can think of opacity as the strength of the blend effect between the BG Color and the FG Color.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Scale (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Scale receives its X, Y, Z, scale values either by entering them into the Scale tab in the Edit Panel or by connecting an input to Scale on the node in the Workspace.

Scale is the scale of the texture pattern that will be applied to your object surface.

If the World Coordinates checkbox is checked in the edit panel then Scale defines the scale of the texture in relation to the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, and Z respectively.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures scale is derived from the selected Reference Object keyframed Scale values.

Connections made in the workspace area will override any values entered by the user into the edit panel.

#### Position (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Position receives its X, Y, Z, coordinate information either by entering it into the Position tab in the Edit Panel or by connecting an input to the Position connection on the node in the Workspace Area.

Position defines the center coordinate of the texture. By default these are offset values of the objects coordinates also sometimes referred to as the objects pivot point.

If the World Coordinates checkbox is checked in the Edit Panel then Position defines the center of the texture as offset values from the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, Z format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures position is derived from the selected Reference Objects keyframed Position values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

#### Rotation (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Rotation receives H (Heading), P (Pitch), and B (Bank) rotational information either by entering degrees into the Rotation tab in the Edit Panel or as radians by connecting an input to the Rotation connection on the node in the Workspace Area.

Rotation defines the axial rotation around the textures center point. By default these are offset values of the objects a rotational settings.

If the World Coordinates checkbox is checked in the Edit Panel then Rotation values define the texture rotation as offset values from the world coordinates of your scene which should be fixed. You can rely on your world coordinate rotation values to always be 0,0,0, in H, P, B format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures rotation is derived from the selected Reference Objects keyframed Rotation values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.



## Outputs

### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

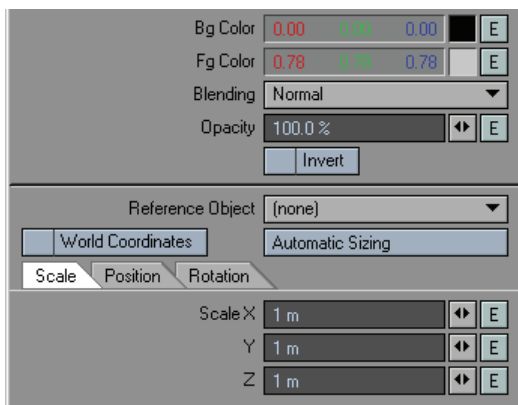
### Alpha (type: scalar):

Outputs single channel alpha (grey) information evaluated on a per spot basis.

### Bump (type: vector):

Outputs component vector direction information the length of which are the product of the bump amplitude value.

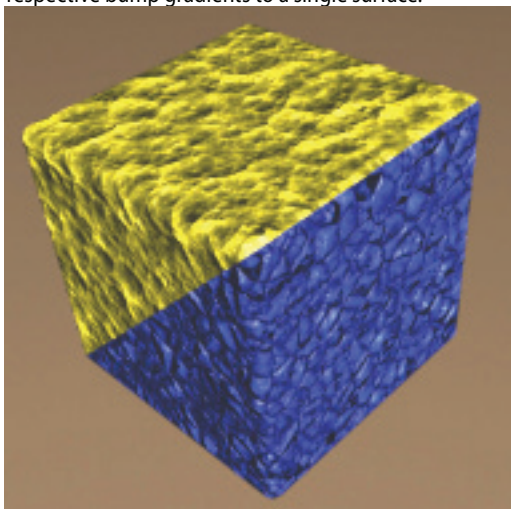
## Edit Panel



Reference Object, World Coordinates and Automatic Sizing are features exclusive to this node's Edit Panel.

## Usage Example

Using checkerboard to apply two textures and their respective bump gradients to a single surface.



This example uses basically the same principles discussed in the Bricks Usage Example section that precedes the Checkerboard node description here.



Of special interest in this node network is how the bump maps from two completely different textures are masked off and then added together for the final result. The masking procedure is done in the same way we discussed earlier using background and foreground color Values to control the bump amplitude of in this case, two individual textures zeroing them out where they are not wanted.

Adding the two separate bump maps together is as simple as adding a Math -> Vector -> Add node and connecting the two respective bump outputs to the A and B inputs.

See the 3D Checkerboard Example preset in the Manual\_Examples library for the Surface Preset shelf [F8] for examination and specific node settings.



## Crackle

### Description

Cellular type texture good for amphibian type skin, scabs, rocks, gravel, asphalt, and etc.

### Inputs

#### Bg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the background color layer.

#### Fg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the foreground color layer.

#### Blending (type: integer):

Specifies one of the blending types (0~13) found in the blending type list at the beginning of this chapter.

Can receive input from other nodes in the network or can be specified to by selecting a blending mode from the Blending pull-down menu in the Edit Panel for the node.

For general use specifying the blending mode by using the Blending pull-down menu in the Edit Panel will probably be the most desirable method of use. However, for larger networks where control of the blending modes of many texture nodes simultaneously is needed or when conditional blending is desired then using a node connection to control this value may be advantageous.

#### Opacity (type: scalar):

Opacity is the amount that the selected blending mode is applied. You can think of opacity as the strength of the blend effect between the BG Color and the FG Color.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Small Scale (type: scalar):

Like Lacunarity, Small Scale is a counterpart to the fractal dimension that describes the texture of a fractal. It has to do with the size distribution of the gaps between detail levels. The Small Scale value sets the amount of change in scale between successive levels of detail.

If you're seeing repeating patterns in the texture it might be a good idea to alter the Small Scale value for the texture.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify values for this attribute using the controls available in the Edit Panel for this node.

#### Frequencies (type: scalar):

"Frequencies" as used in this node is really just another name for octaves as defined in some of the descriptions of fractal type textures within this document.

You can think of "Frequencies" as the number of levels of detail mentioned in the discussion just above for Small Scale. At one "Frequency", only the basic pattern is used. At 2 "Frequencies" given a Small Scale value of 2, a one half scale pattern is overlaid on top of the basic pattern. And at three Frequencies, the overlays are one quarter scale over the one half scale over the basic and so on.

If Small Scale were 3 in this example, it would mean that each new iteration would be one third the scale of the previous layer.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify the values for this attribute using the controls available in the Edit Panel for this node.

#### Function (type: function):

Connecting a function node to this input will transform the texture value of this texture based on the graph defined by the function thereby extending the range of this nodes producible patterns.

See the manual section on Function nodes for a description of how texture values can be transformed.

Can receive inputs from other function nodes in the network only.

#### Bump (type: vector):

Specifies a vector direction for modifying the surface normal. This is surface normal direction information and affects the way the surface is shaded. Care should be taken when connecting to this input. Connecting dissimilar types or non directional vectors may cause the surface to shade wrongly.

Can receive input from other nodes in the network only.





## Bump Amplitude (type: scalar):

Specifies the bump height or "amplitude" of the Bump directional vectors.

Using Alpha or luminance (connecting Color) values from other textures in the network along with their respective bumps can have the effect of blending the bump textures together.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Since Bump Amplitude is a shading feature and not a displacement coordinate system value it is specified as a percentage. Specifically as a percentage of the texture value.

## Scale (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Scale receives its X, Y, Z, scale values either by entering them into the Scale tab in the Edit Panel or by connecting an input to Scale on the node in the Workspace.

Scale is the scale of the texture pattern that will be applied to your object surface.

If the World Coordinates checkbox is checked in the edit panel then Scale defines the scale of the texture in relation to the world coordinates of your scene. Also known as world coordinate 0,0,0, in X, Y, and Z respectively.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures scale is derived from the selected Reference Object keyframed Scale values.

Connections made in the workspace area will override any values entered by the user into the edit panel.

## Position (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Position receives its X, Y, Z, coordinate information either by entering it into the Position tab in the Edit Panel or by connecting an input to the Position connection on the node in the Workspace Area.

Position defines the center coordinate of the texture. By default these are offset values of the objects coordinates also sometimes referred to as the objects pivot point.

If the World Coordinates checkbox is checked in the Edit Panel then Position defines the center of the texture as offset values from the world coordinates of your scene. Also known as world coordinate 0,0,0, in X, Y, Z format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures position is derived from the selected Reference Objects keyframed Position values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Rotation (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Rotation receives H (Heading), P (Pitch), and B (Bank) rotational information either by entering degrees into the Rotation tab in the Edit Panel or as radians by connecting an input to the Rotation connection on the node in the Workspace Area.

Rotation defines the axial rotation around the textures center point. By default these are offset values of the objects a rotational settings.

If the World Coordinates checkbox is checked in the Edit Panel then Rotation values define the texture rotation as offset values from the world coordinates of your scene which should be fixed. You can rely on your world coordinate rotation values to always be 0,0,0, in H, P, B format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures rotation is derived from the selected Reference Objects keyframed Rotation values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Outputs

### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

### Alpha (type: scalar):

Outputs single channel alpha (grey) information evaluated on a per spot basis.

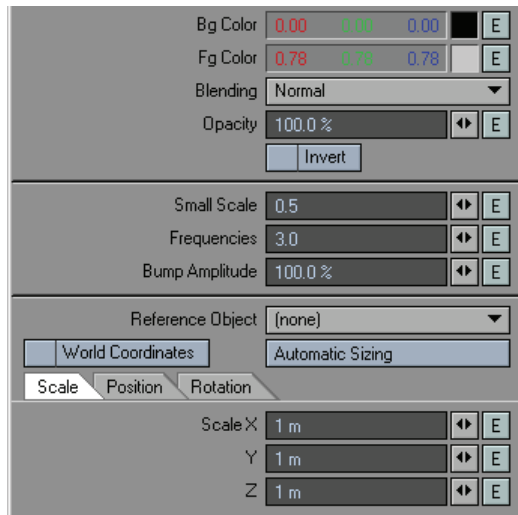
### Bump (type: vector):

Outputs component vector direction information the length of which are the product of the bump amplitude value.





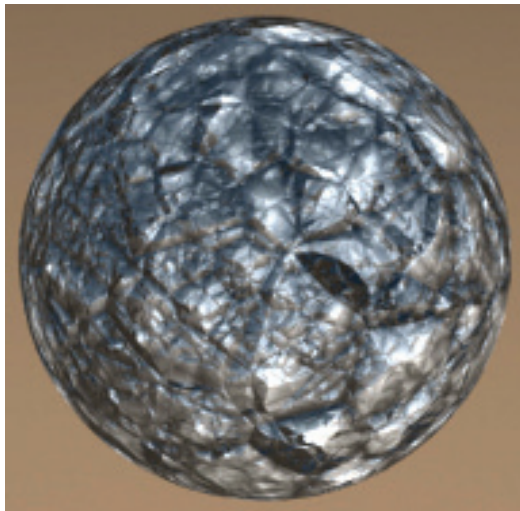
## Edit Panel



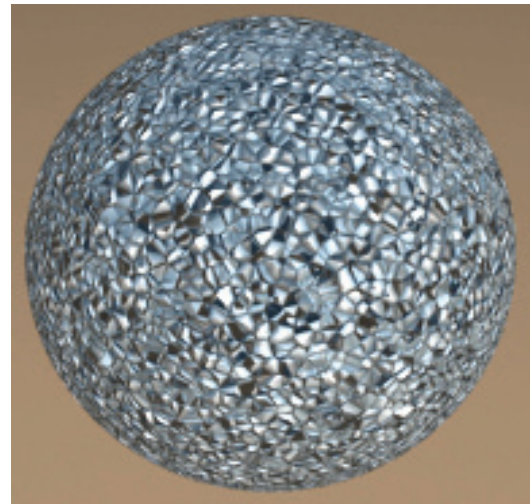
Reference Object, World Coordinates and Automatic Sizing are features exclusive to this nodes Edit Pane. Please see the explanation for these items in the chapter that covers texturing with the Surface Editor.

## Usage Example

Using Crackle for rough rocks, gravel and asphalt.



Small Scale = 0.7 Frequencies = 3

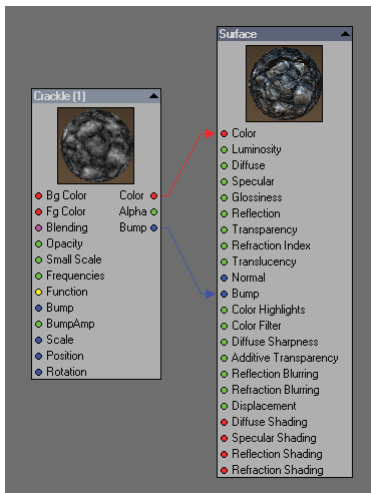


Small Scale = 5 Frequencies = 3



Small Scale = 5 Frequencies = 4.5

As you can see from the images increasing the Small Scale value affects the granularity and general roughness this texture produces. Likewise increasing both the Frequencies value and the Small Scale value produces even more dramatic results along the same lines.



As you can see from the network (above) the example images are the result of adding just a single Crackle node and connecting its Bump output to the Surface destination nodes Bump input and connecting the Color to the Color of the same.

## Crumple

### Description

A cellular type texture good for a wide variety of uses including oceans, windy lake surfaces, to various types of rock, certain types of window glass, and etc.

### Inputs

#### Bg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the background color layer.

#### Fg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the foreground color layer.

#### Blending (type: integer):

Specifies one of the blending types (0~13) found in the blending type list at the beginning of this chapter.

Can receive input from other nodes in the network or can be specified to by selecting a blending mode from the Blending pull-down menu in the Edit Panel for the node.

For general use specifying the blending mode by using the Blending pull-down menu in the Edit Panel will probably be the most desirable method of use. However, for larger networks where control of the blending modes of many texture nodes simultaneously is needed or when conditional blending is desired then using a node connection to control this value may be advantageous.

#### Opacity (type: scalar):

Opacity is the amount that the selected blending mode is applied. You can think of opacity as the strength of the blend effect between the BG Color and the FG Color.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Small Scale (type: scalar):

Like Lacunarity, Small Scale is a counterpart to the fractal dimension that describes the texture of a fractal. It has to do with the size distribution of the gaps between detail levels. The Small Scale value sets the amount of change in scale between successive levels of detail.



If you're seeing repeating patterns in the texture it might be a good idea to alter the Small Scale value for the texture.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify values for this attribute using the controls available in the Edit Panel for this node.

## Frequencies (type: scalar):

"Frequencies" as used in this node is really just another name for octaves as defined in some of the descriptions of fractal type textures within this document.

You can think of "Frequencies" as the number of levels of detail mentioned in the discussion just above for Small Scale. At one "Frequency", only the basic pattern is used. At 2 "Frequencies" given a Small Scale value of 2, a one half scale pattern is overlaid on top of the basic pattern. And at three Frequencies, the overlays are one quarter scale over the one half scale over the basic and so on.

If Small Scale were 3 in this example, it would mean that each new iteration would be one third the scale of the previous layer.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify the values for this attribute using the controls available in the Edit Panel for this node.

## Function (type: function):

Connecting a function node to this input will transform the texture value of this texture based on the graph defined by the function thereby extending the range of this nodes producible patterns.

See the manual section on Function nodes for a description of how texture values can be transformed.

Can receive inputs from other function nodes in the network only.

## Bump (type: vector):

Specifies a vector direction for modifying the surface normal. This is surface normal direction information and affects the way the surface is shaded. Care should be taken when connecting to this input. Connecting dissimilar types or non directional vectors may cause the surface to shade wrongly.

Can receive input from other nodes in the network only.

## Bump Amplitude (type: scalar):

Specifies the bump height or "amplitude" of the Bump directional vectors.

Using Alpha or luminance (connecting Color) values from other textures in the network along with their respective bumps can have the effect of blending the bump textures together.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Since Bump Amplitude is a shading feature and not a displacement coordinate system value it is specified as a percentage. Specifically as a percentage of the texture value.

## Scale (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Scale receives its X, Y, Z, scale values either by entering them into the Scale tab in the Edit Panel or by connecting an input to Scale on the node in the Workspace.

Scale is the scale of the texture pattern that will be applied to your object surface.

If the World Coordinates checkbox is checked in the edit panel then Scale defines the scale of the texture in relation to the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, and Z respectively.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures scale is derived from the selected Reference Object keyframed Scale values.

Connections made in the workspace area will override any values entered by the user into the edit panel.

## Position (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Position receives its X, Y, Z, coordinate information either by entering it into the Position tab in the Edit Panel or by connecting an input to the Position connection on the node in the Workspace Area.

Position defines the center coordinate of the texture. By default these are offset values of the objects coordinates also sometimes referred to as the objects pivot point.

If the World Coordinates checkbox is checked in the Edit Panel then Position defines the center of the texture as offset values from the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, Z format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures position is derived from the selected Reference Objects keyframed Position values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.



## Rotation (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Rotation receives H (Heading), P (Pitch), and B (Bank) rotational information either by entering degrees into the Rotation tab in the Edit Panel or as radians by connecting an input to the Rotation connection on the node in the Workspace Area.

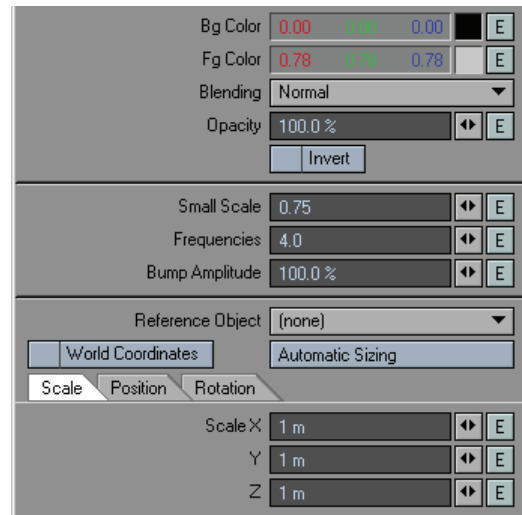
Rotation defines the axial rotation around the textures center point. By default these are offset values of the objects a rotational settings.

If the World Coordinates checkbox is checked in the Edit Panel then Rotation values define the texture rotation as offset values from the world coordinates of your scene which should be fixed. You can rely on your world coordinate rotation values to always be 0,0,0, in H, P, B format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures rotation is derived from the selected Reference Objects keyframed Rotation values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Edit Panel



Reference Object, World Coordinates and Automatic Sizing are features exclusive to this nodes Edit Panel. Please see the explanation for these items in the chapter that covers texturing with the Surface Editor.

## Outputs

### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

### Alpha (type: scalar):

Outputs single channel alpha (grey) information evaluated on a per spot basis.

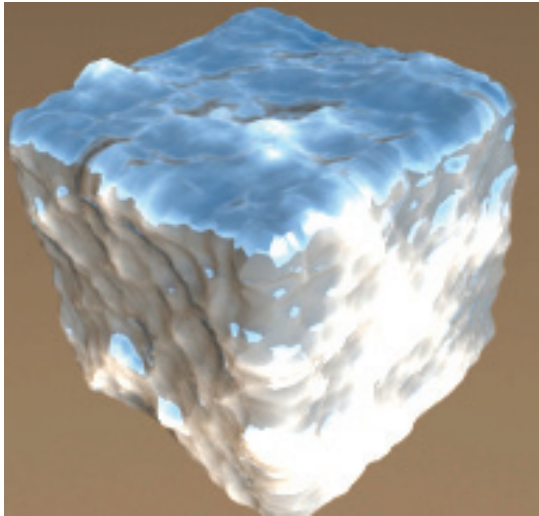
### Bump (type: vector):

Outputs component vector direction information the length of which are the product of the bump amplitude value.



## Usage Example

Roughing out the start of an ocean texture using crumple - Animated.

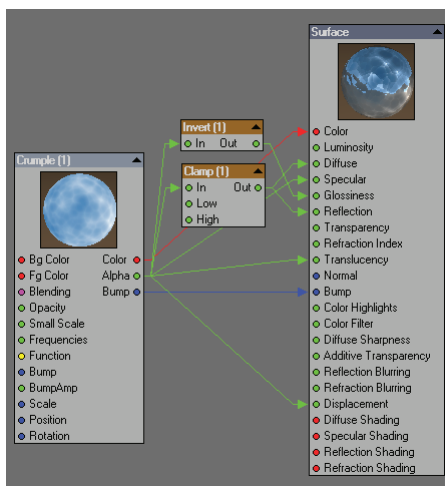


By using this texture with its default values on the Color and Displacement channels a simple but not entirely unconvincing ocean surface can be produced.



**NOTE:** Anytime the Surface destination node's Displacement input is used you must remember to open the Object Properties panel for the object that the surface is applied to, check the Enable Bump checkbox in the Deform tab, and enter a Distance value which specifies the amount of displacement you wish to incur.

Using the same Crumple node to define some of the other surface properties will enhance this simple ocean considerably.



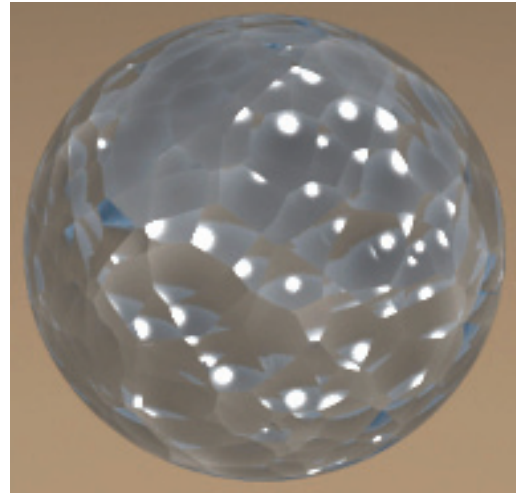
Here as you can see, we are using its Alpha output directly to define the Specular and Translucent channels besides the Displacement. Additionally we are using an inverted Alpha from the same Crumple node to define the Glossiness of the surface. By using an inverted Alpha to define the glossiness we are saying in this case, to make the peaks of the waves low gloss and the bottom of the waves high gloss.

When this effect is combined with high specular values for the peaks and low specular values for the bottoms of the waves it creates an iridescent look to the surface when close and a windblown white-cap look when far away.

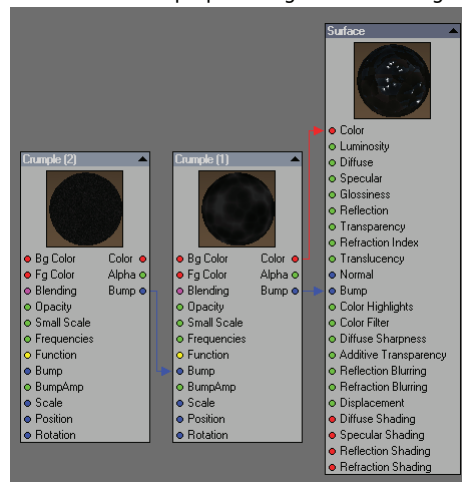
We are also using clamped Alpha values to define diffuse and reflection. Please see the description for the clamp node to get an idea of what's going on here. But basically what we're doing by employing the use of a Clamp node is restricting the Alpha channel from reducing the Reflectivity and Diffuse levels any lower than 0.5; the low value entered into the Clamp node Edit Panel

This example is animated for you so try rendering a few frames to see how texture movement can produce what looks like the waves of an ocean surface.

Getting that chipped obsidian look with Crumple:



Reducing the number of frequencies in Crumple (1) to 1.0 produces a texture pattern that looks very much like chipped obsidian with the proper background and foreground colors.



Adding another instance of crumple produces "Micro-Bump" effects that help the effect and break up the specular highlights much like we would expect from actual obsidian.

This "Micro-bump" effect is a common texturing technique and is achieved by scaling down the texture in relation to both the surface Scale of the texture that it is being applied to and the general dimensions of the camera view plane, and then additionally reducing the overall amplitude of the affecting bump.

In this example with the size of the camera view plane just a little over one meter at object center and the Scale of the larger texture being one meter, the Scale for the "Micro-bump" was set to 10mm for X,Y, and Z. and the Bump Amplitude was set to 10%.

See the "3D Crumple\_Example 2" surface preset in the Surface Preset [F8] shelf that was installed with your LightWave3D content.





## Crust

### Description

A cellular type texture that produces intersecting circular patterns and provides controls for transition gradients between the circular foreground and background colors.

### Inputs

#### Bg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the background color layer.

#### Fg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the foreground color layer.

#### Blending (type: integer):

Specifies one of the blending types (0~13) found in the blending type list at the beginning of this chapter.

Can receive input from other nodes in the network or can be specified to by selecting a blending mode from the Blending pull-down menu in the Edit Panel for the node.

For general use specifying the blending mode by using the Blending pull-down menu in the Edit Panel will probably be the most desirable method of use. However, for larger networks where control of the blending modes of many texture nodes simultaneously is needed or when conditional blending is desired then using a node connection to control this value may be advantageous.

#### Opacity (type: scalar):

Opacity is the amount that the selected blending mode is applied. You can think of opacity as the strength of the blend effect between the BG Color and the FG Color.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Width (type: scalar):

The Width value specified as a percentage, controls the width or "diameter" of the ball shapes that produce a circular look where an object surface intersects with the texture.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this value as a percentage using the controls available in the Edit Panel for this node.

#### Contrast (type: scalar):

Controls the contrast between the foreground and background. High values produce very sharp transitions between the foreground and background. Low values create smooth gradient transitions between the foreground and the background.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this value as a percentage using the controls available in the Edit Panel for this node.

#### Function (type: function):

Connecting a function node to this input will transform the texture value of this texture based on the graph defined by the function thereby extending the range of this nodes producible patterns.

See the manual section on Function nodes for a description of how texture values can be transformed.

Can receive inputs from other function nodes in the network only.

#### Bump (type: vector):

Specifies a vector direction for modifying the surface normal. This is surface normal direction information and affects the way the surface is shaded. Care should be taken when connecting to this input. Connecting dissimilar types or non directional vectors may cause the surface to shade wrongly.

Can receive input from other nodes in the network only.

#### Bump Amplitude (type: scalar):

Specifies the bump height or "amplitude" of the Bump directional vectors.

Using Alpha or luminance (connecting Color) values from other textures in the network along with their respective bumps can have the effect of blending the bump textures together.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Since Bump Amplitude is a shading feature and not a displacement coordinate system value it is specified as a percentage. Specifically as a percentage of the texture value.





## Scale (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Scale receives its X, Y, Z, scale values either by entering them into the Scale tab in the Edit Panel or by connecting an input to Scale on the node in the Workspace.

Scale is the scale of the texture pattern that will be applied to your object surface.

If the World Coordinates checkbox is checked in the edit panel then Scale defines the scale of the texture in relation to the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, and Z respectively.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures scale is derived from the selected Reference Object keyframed Scale values.

Connections made in the workspace area will override any values entered by the user into the edit panel.

## Position (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Position receives its X, Y, Z, coordinate information either by entering it into the Position tab in the Edit Panel or by connecting an input to the Position connection on the node in the Workspace Area.

Position defines the center coordinate of the texture. By default these are offset values of the objects coordinates also sometimes referred to as the objects pivot point.

If the World Coordinates checkbox is checked in the Edit Panel then Position defines the center of the texture as offset values from the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, Z format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures position is derived from the selected Reference Objects keyframed Position values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Rotation (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Rotation receives H (Heading), P (Pitch), and B (Bank) rotational information either by entering degrees into the Rotation tab in the Edit Panel or as radians by connecting an input to the Rotation connection on the node in the Workspace Area.

Rotation defines the axial rotation around the textures center point. By default these are offset values of the objects a rotational settings.

If the World Coordinates checkbox is checked in the Edit Panel then Rotation values define the texture rotation as offset values from the world coordinates of your scene which should be fixed. You can rely on your world coordinate rotation values to always be 0,0,0, in H, P, B format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures rotation is derived from the selected Reference Objects keyframed Rotation values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Outputs

### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

### Alpha (type: scalar):

Outputs single channel alpha (grey) information evaluated on a per spot basis.

### Bump (type: vector):

Outputs component vector direction information the length of which are the product of the bump amplitude value.

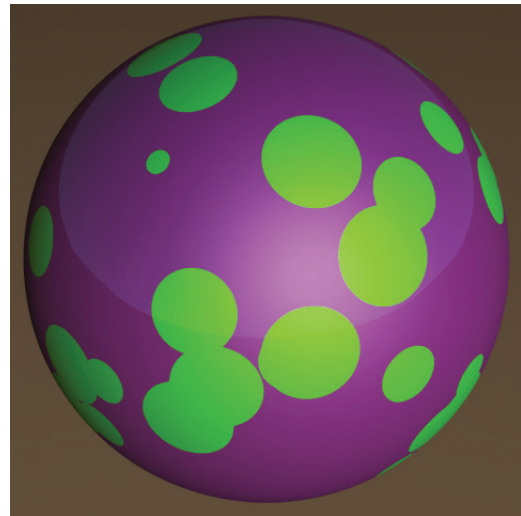


## Edit Panel

Reference Object, World Coordinates and Automatic Sizing are features exclusive to this nodes Edit Panel. Please see the explanation for these items in the chapter that covers texturing with the Surface Editor.

## Usage Example

The random dottedness of the Crust texture.

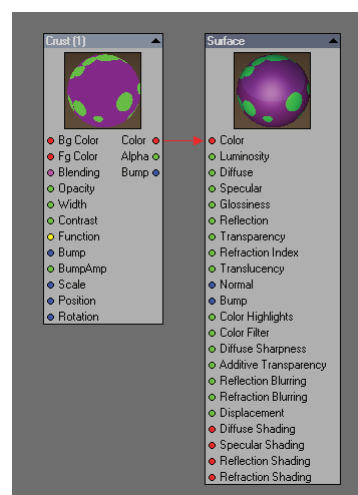


Simply by adding a Crust node, increasing its Contrast percentage value to 100%, setting its Width percentage value to 50%, and connecting its Color output to the Surface destination nodes' Color input we can get a good idea of the basic nature of the Crust texture.

One aspect of Crust that is not apparently evident is that the filled circles (green in this example) are actually spherical and the circles you see produced in the renderer above, are produced where the object surface intersects or passes through, these spheres.

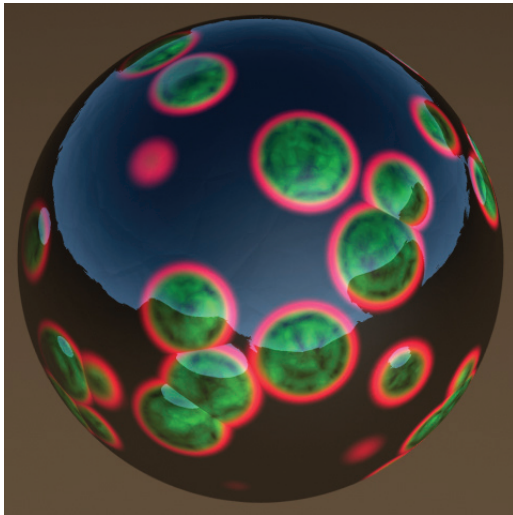
Within the texture these "spheres" can intersect with each other and globulate together. With much higher Width percentage values and low Contrast this globulation can produce a very different overall pattern.

Here is the node network for the renderer example adjacent left.

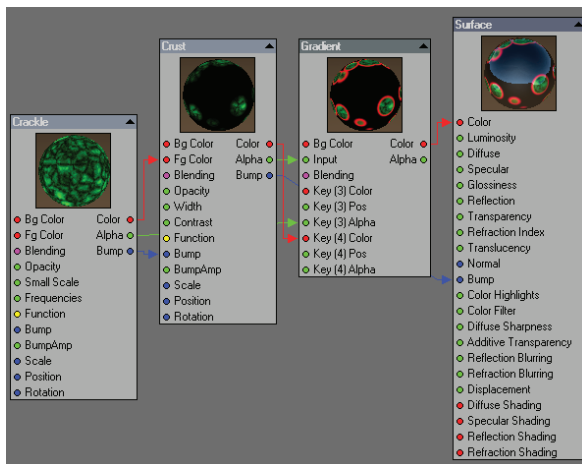




Alien pond eggs using the Crust texture and a Gradient node. - Animated.



The simplicity of the Crust surface offers a special opportunity to take a look at how Gradient Input connections work and how Key Alpha and Key Color can be used.



Take a look at this network example.

The most notable connection to observe is the Crust nodes' Alpha output to the Gradient nodes Input input. When you connect a texture in this way to the Gradient nodes Input basically the gradient is conforming or shaping to the texture values being used. In this case circular gradients are produced based on the circularity of the Crust texture values.

To see this exemplified double click on the "3D Crust example 2" preset in the "Manual\_Examples" library of the Surface Preset shelf [F8] and disconnect all other inputs from the gradient except for the Input that is connected in this example. Double click on the Gradient node to view the Key Color spreads that were set up for this example. An immediate correlation should be apparent.

Connecting the Color output from the Crust node to any of the Key Colors displayed will replace that Key Color definition defined in the edit panel for the Gradient, with the Crust texture.

Connecting the Alpha output from the Crackle node to any of the Key Alpha inputs will replace the

gradient Alpha with the textured Alpha values from the Crackle node. To see this exemplified load the preset as we described above, and disconnect and reconnect only the Crackle Alpha output from the Key [3] Alpha input on the Gradient node.

Additionally the Bump channels from Crackle and Crust are mixed together as we described previously in this manual to complete the surface attributes that might look something like alien pond eggs.

This example was created to animate so go ahead and select Make Preview in the Preview pull down menu located in the VIPER window.



## Dots

### Description

Procedurally generated three dimensional grid of ball-like texture elements.

### Inputs

#### Bg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the background color layer.

#### Fg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the foreground color layer.

#### Blending (type: integer):

Specifies one of the blending types (0~13) found in the blending type list at the beginning of this chapter.

Can receive input from other nodes in the network or can be specified to by selecting a blending mode from the Blending pull-down menu in the Edit Panel for the node.

For general use specifying the blending mode by using the Blending pull-down menu in the Edit Panel will probably be the most desirable method of use. However, for larger networks where control of the blending modes of many texture nodes simultaneously is needed or when conditional blending is desired then using a node connection to control this value may be advantageous.

#### Opacity (type: scalar):

Opacity is the amount that the selected blending mode is applied. You can think of opacity as the strength of the blend effect between the BG Color and the FG Color.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Diameter (type: scalar):

The Diameter value specified as a percentage, controls the "width" or diameter of the ball shapes that produce a circular look where an object surface intersects with the texture.

Since the sphere-like texture elements are aligned and spaced on a cubic grid by default it is entirely possible to apply this texture to a box or flat surface and see no results at all. What's happening in this circumstance is that the balls are aligning on either side of the polygon surface. The surface plane essentially

is passing through a part of the texture that contains no variant values - the space between the balls - and thus no visual results are produced.

In this situation increasing the Diameter parameter to between 120% and 150% will allow you to see the results of the texture in the VIPER preview in order to interactively Position the texture at the optimal offset values.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this value as a percentage using the controls available in the Edit Panel for this node.

#### Contrast (type: scalar):

Controls the contrast between the foreground and background. High values produce very sharp transitions between the foreground and background. Low values create smooth gradient transitions between the foreground and the background.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this value as a percentage using the controls available in the Edit Panel for this node.

#### Function (type: function):

Connecting a function node to this input will transform the texture value of this texture based on the graph defined by the function thereby extending the range of this nodes producible patterns.

See the manual section on Function nodes for a description of how texture values can be transformed.

Can receive inputs from other function nodes in the network only.

#### Bump (type: vector):

Specifies a vector direction for modifying the surface normal. This is surface normal direction information and affects the way the surface is shaded. Care should be taken when connecting to this input. Connecting dissimilar types or non directional vectors may cause the surface to shade wrongly.

Can receive input from other nodes in the network only.

#### Bump Amplitude (type: scalar):

Specifies the bump height or "amplitude" of the Bump directional vectors.

Using Alpha or luminance (connecting Color) values from other textures in the network along with their respective bumps can have the effect of blending the bump textures together.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Since Bump Amplitude is a shading feature and not a displacement coordinate system value it is specified as a percentage. Specifically as a percentage of the texture value.

#### Scale (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Scale receives its X, Y, Z, scale values either by entering them into



the Scale tab in the Edit Panel or by connecting an input to Scale on the node in the Workspace.

Scale is the scale of the texture pattern that will be applied to your object surface.

If the World Coordinates checkbox is checked in the edit panel then Scale defines the scale of the texture in relation to the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, and Z respectively.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures scale is derived from the selected Reference Object keyframed Scale values.

Connections made in the workspace area will override any values entered by the user into the edit panel.

## Position (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Position receives its X, Y, Z. coordinate information either by entering it into the Position tab in the Edit Panel or by connecting an input to the Position connection on the node in the Workspace Area.

Position defines the center coordinate of the texture. By default these are offset values of the objects coordinates also sometimes referred to as the objects pivot point.

If the World Coordinates checkbox is checked in the Edit Panel then Position defines the center of the texture as offset values from the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, Z format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures position is derived from the selected Reference Objects keyframed Position values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Rotation (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Rotation receives H (Heading), P (Pitch), and B (Bank) rotational information either by entering degrees into the Rotation tab in the Edit Panel or as radians by connecting an input to the Rotation connection on the node in the Workspace Area.

Rotation defines the axial rotation around the textures center point. By default these are offset values of the objects a rotational settings.

If the World Coordinates checkbox is checked in the Edit Panel then Rotation values define the texture rotation as offset values from the world coordinates of your scene which should be fixed.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures rotation is derived from the selected Reference Objects keyframed Rotation values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Outputs

### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

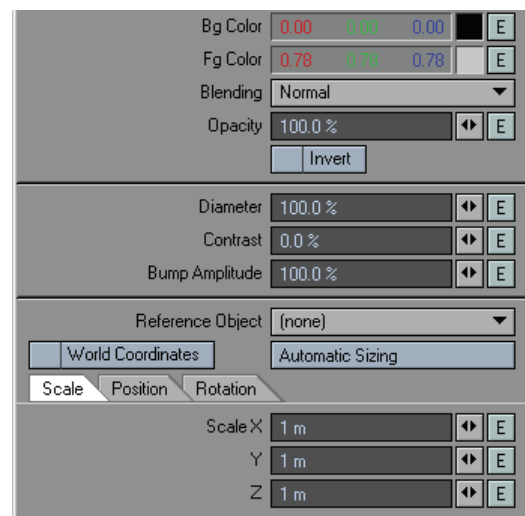
### Alpha (type: scalar):

Outputs single channel alpha (grey) information evaluated on a per spot basis.

### Bump (type: vector):

Outputs component vector direction information the length of which are the product of the bump amplitude value.

## Edit Panel



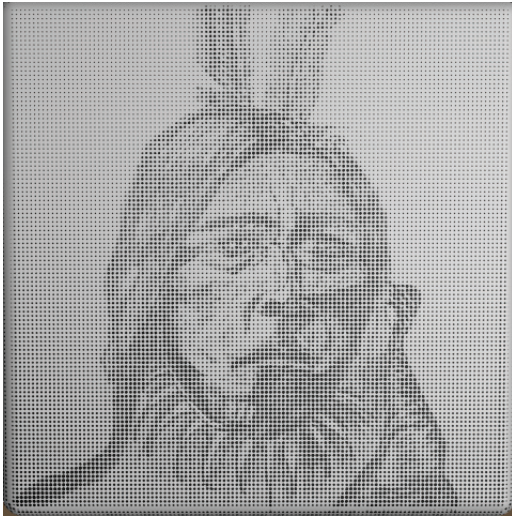
Reference Object, World Coordinates and Automatic Sizing are features exclusive to this nodes Edit Panel. Please see the explanation for these items in the chapter that covers texturing with the Surface Editor.



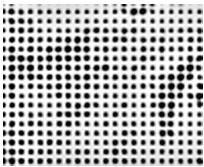


## Usage Example

Create dot matrix or newsprint like renderings from any image using the Dots.node.

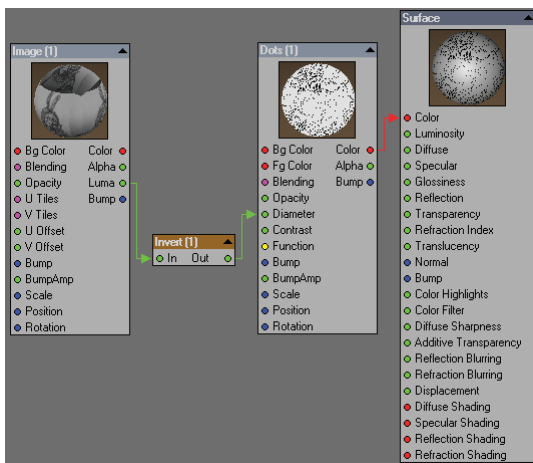


As you can see from the above example and the closeup section below, connecting the Luma output from an Image node to the Dot Diameter input produces an effect very close to newsprint printing dot density patterns.



This effect can be used for color or black and white but here in this example is being defined by the foreground and background colors of the Dot node.

Here is the node network for the example. Other procedural texture nodes can be used in the same way as the Image node to control dot diameter.



The invert node in the network above is being used to invert the luminance values of the loaded image. Luminance values of 100% (white) would result in a Diameter value of 100% and likewise 0% for black.

For this affect to work properly we want the opposite influence to occur from the luminance values. The invert node inverts ranges from zero to one and thus is just what we need to make the dot diameters larger for darker areas and smaller for lighter areas.

## FBM

### Description

A fractional Brownian motion (FBM) fractal texture. The results of several scientific studies have shown that many real world natural textures have the same kind of spectra as produced by FBM making it a natural choice for various kinds of naturally occurring textures and patterns.

### Inputs

#### Bg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the background color layer.

#### Fg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the foreground color layer.

#### Blending (type: integer):

Specifies one of the blending types (0~13) found in the blending type list at the beginning of this chapter.

Can receive input from other nodes in the network or can be specified to by selecting a blending mode from the Blending pull-down menu in the Edit Panel for the node.

For general use specifying the blending mode by using the Blending pull-down menu in the Edit Panel will probably be the most desirable method of use. However, for larger networks where control of the blending modes of many texture nodes simultaneously is needed or when conditional blending is desired then using a node connection to control this value may be advantageous.

#### Opacity (type: scalar):

Opacity is the amount that the selected blending mode is applied. You can think of opacity as the strength of the blend effect between the BG Color and the FG Color.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.





## Small Scale (type: scalar):

Like Lacunarity, Small Scale is a counterpart to the fractal dimension that describes the texture of a fractal. It has to do with the size distribution of the gaps between detail levels. The Small Scale value sets the amount of change in scale between successive levels of detail.

If you're seeing repeating patterns in the texture it might be a good idea to alter the Small Scale value for the texture.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify values for this attribute using the controls available in the Edit Panel for this node.

## Contrast (type: scalar):

Sets the contrast of the texture value gradient. Higher values produce less gradient information between texture value extremes and therefore less noise is produced. The lower values spread the gradient more evenly between the extremes of the texture values and thus allow for more overall noise in the final output from this node

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify the values for this attribute using the controls available in the Edit Panel for this node.

## Frequencies (type: scalar):

"Frequencies" as used in this node is really just another name for octaves as defined in some of the descriptions of fractal type textures within this document.

You can think of "Frequencies" as the number of levels of detail mentioned in the discussion just above for Small Scale. At one "Frequency", only the basic pattern is used. At 2 "Frequencies" given a Small Scale value of 2, a one half scale pattern is overlaid on top of the basic pattern. And at three Frequencies, the overlays are one quarter scale over the one half scale over the basic and so on.

If Small Scale were 3 in this example, it would mean that each new iteration would be one third the scale of the previous layer.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify the values for this attribute using the controls available in the Edit Panel for this node.

## Function (type: function):

Connecting a function node to this input will transform the texture value of this texture based on the graph defined by the function thereby extending the range of this nodes producible patterns.

See the manual section on Function nodes for a description of how texture values can be transformed.

Can receive inputs from other function nodes in the network only.

## Bump (type: vector):

Specifies a vector direction for modifying the surface normal. This is surface normal direction information and affects the way the surface is shaded.

Care should be taken when connecting to this input. Connecting dissimilar types or non directional vectors may cause the surface to shade wrongly.

Can receive input from other nodes in the network only.

## Bump Amplitude (type: scalar):

Specifies the bump height or "amplitude" of the Bump directional vectors.

Using Alpha or luminance (connecting Color) values from other textures in the network along with their respective bumps can have the effect of blending the bump textures together.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Since Bump Amplitude is a shading feature and not a displacement coordinate system value it is specified as a percentage. Specifically as a percentage of the texture value.

## Scale (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Scale receives its X, Y, Z, scale values either by entering them into the Scale tab in the Edit Panel or by connecting an input to Scale on the node in the Workspace.

Scale is the scale of the texture pattern that will be applied to your object surface.

If the World Coordinates checkbox is checked in the edit panel then Scale defines the scale of the texture in relation to the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, and Z respectively.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures scale is derived from the selected Reference Object keyframed Scale values.

Connections made in the workspace area will override any values entered by the user into the edit panel.

## Position (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Position receives its X, Y, Z, coordinate information either by entering it into the Position tab in the Edit Panel or by connecting an input to the Position connection on the node in the Workspace Area.

Position defines the center coordinate of the texture. By default these are offset values of the objects coordinates also sometimes



referred to as the objects pivot point.

If the World Coordinates checkbox is checked in the Edit Panel then Position defines the center of the texture as offset values from the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, Z format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures position is derived from the selected Reference Objects keyframed Position values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Rotation (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Rotation receives H (Heading), P (Pitch), and B (Bank) rotational information either by entering degrees into the Rotation tab in the Edit Panel or as radians by connecting an input to the Rotation connection on the node in the Workspace Area.

Rotation defines the axial rotation around the textures center point. By default these are offset values of the objects a rotational settings.

If the World Coordinates checkbox is checked in the Edit Panel then Rotation values define the texture rotation as offset values from the world coordinates of your scene which should be fixed. You can rely on your world coordinate rotation values to always be 0,0,0, in H, P, B format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures rotation is derived from the selected Reference Objects keyframed Rotation values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Outputs

### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

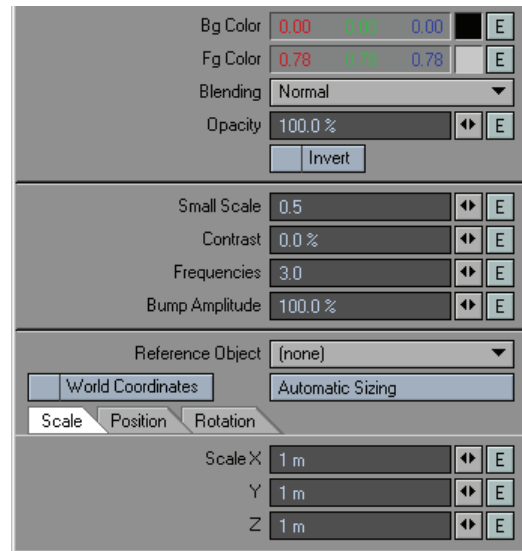
### Alpha (type: scalar):

Outputs single channel alpha (grey) information evaluated on a per spot basis.

### Bump (type: vector):

Outputs component vector direction information the length of which are the product of the bump amplitude value.

## Edit Panel

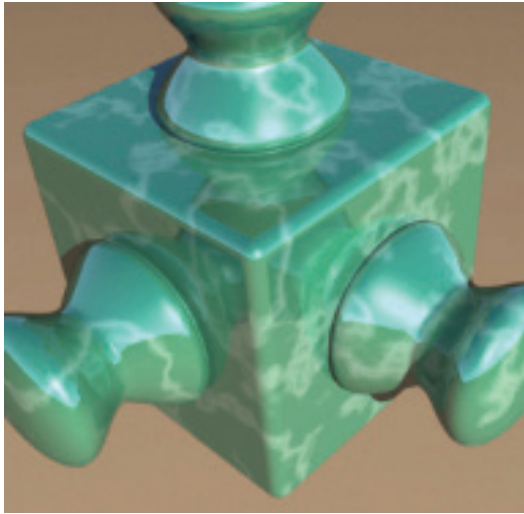


Reference Object, World Coordinates and Automatic Sizing are features exclusive to this nodes Edit Panel. Please see the explanation for these items in the chapter that covers texturing with the Surface Editor.

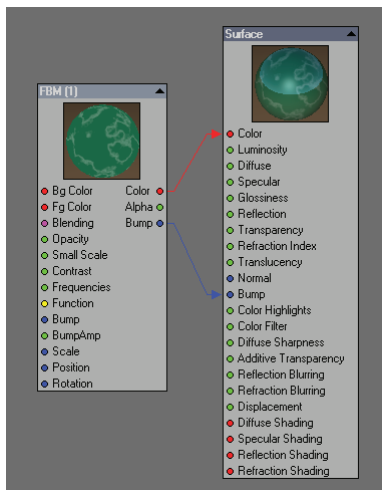


## Usage Example

FBM, A better marble than marble?



As you can see FBM can produce quite convincing marble and natural rock texture patterns.



This ultra-simple node network demonstrates how.

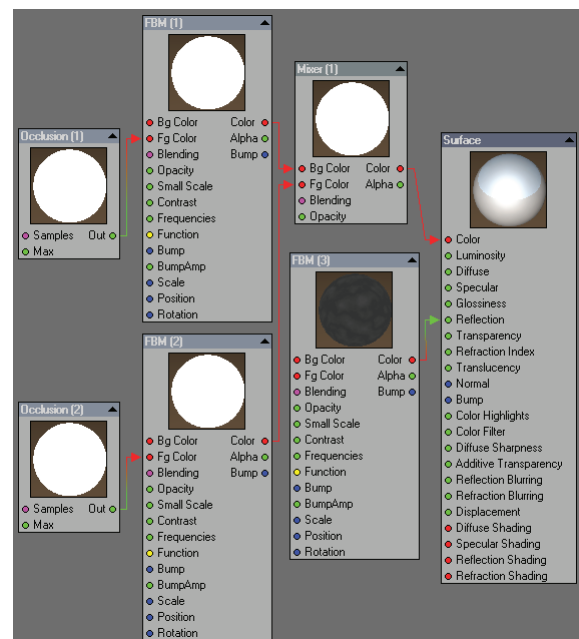
Creating soot and dirt with FBM and the Occlusion shader.



To understand how this texture is working load the "3D FBM\_Example" preset from the "Manual\_Examples" Surface Preset [F8] library and have a look at the various Edit Node panels. Occlusion shading is being used for two levels of soot mapping. Both Occlusion shaders are being used in Ranged mode. The one being used for the final textures background is set to a maximum range of 12cm and the one being used for the foreground color is set to 5cm. This is intentionally set in relation to the scale of the two FBM nodes being used to pattern the occluded areas. The FBM nodes' Scale values are 10cm and 20cm respectively.

Closely examined, you should be able to detect thicker darker patterns toward the inner areas and thinner wispy patterns toward the outer areas as if the soot buildup was affected by an draft or airflow as it passes through the cracks of the vent area between the base of the stubs and the flat areas of the cube.

The Occlusion shader is extremely useful for adding or eliminating texture detail to concave geometric areas of your objects surface.



Be sure and check out the "3D FBM\_Example" preset from the "Manual\_Examples" Surface Preset [F8] library or load the "3DFBM\_CubeStubby\_2.lwo" object that were installed with your LightWave3D content.



## Grid

### Description

Procedurally generated grid pattern. This grid pattern can vary the thickness of the lines that form the grid and has the ability to define a soft edge falloff from line to center in user specified amounts.

### Inputs

#### Bg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the background color layer.

#### Fg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the foreground color layer.

#### Blending (type: integer):

Specifies one of the blending types (0~13) found in the blending type list at the beginning of this chapter.

Can receive input from other nodes in the network or can be specified to by selecting a blending mode from the Blending pull-down menu in the Edit Panel for the node.

For general use specifying the blending mode by using the Blending pull-down menu in the Edit Panel will probably be the most desirable method of use. However, for larger networks where control of the blending modes of many texture nodes simultaneously is needed or when conditional blending is desired then using a node connection to control this value may be advantageous.

#### Opacity (type: scalar):

Opacity is the amount that the selected blending mode is applied. You can think of opacity as the strength of the blend effect between the BG Color and the FG Color.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Width (type: scalar):

Defines the width of the grid square faces. If you think of this texture in terms of a tile pattern the width parameter defines the size of the tile plates in ratio to the mortar areas or grid lines,

A value of 100 % would be all tile plates (squares) and a value of 0% would be all mortar areas (grid lines) in a mutually exclusive manner.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Contrast (type: scalar):

Controls the contrast amount between the foreground and background. Essentially this acts as a cubic blend or falloff between the background (grid squares) and the foreground (grid lines).

Values near 100% will produce very sharp transitions between grid squares and grid lines. Values near 0% will produce very soft linear gradient transitions between grid squares and grid lines.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Function (type: function):

Connecting a function node to this input will transform the texture value of this texture based on the graph defined by the function thereby extending the range of this nodes producible patterns.

See the manual section on Function nodes for a description of how texture values can be transformed.

Can receive inputs from other function nodes in the network only.

#### Bump (type: vector):

Specifies a vector direction for modifying the surface normal. This is surface normal direction information and affects the way the surface is shaded. Care should be taken when connecting to this input. Connecting dissimilar types or non directional vectors may cause the surface to shade wrongly.

Can receive input from other nodes in the network only.

#### Bump Amplitude (type: scalar):

Specifies the bump height or "amplitude" of the Bump directional vectors.

Using Alpha or luminance (connecting Color) values from other textures in the network along with their respective bumps can have the effect of blending the bump textures together.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Since Bump Amplitude is a shading feature and not a displacement coordinate system value it is specified as a percentage. Specifically as a percentage of the texture value.

#### Scale (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Scale receives its X, Y, Z, scale values either by entering them into the Scale tab in the Edit Panel or by connecting an input to Scale on the node in the Workspace.



Scale is the scale of the texture pattern that will be applied to your object surface.

If the World Coordinates checkbox is checked in the edit panel then Scale defines the scale of the texture in relation to the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, and Z respectively.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures scale is derived from the selected Reference Object keyframed Scale values.

Connections made in the workspace area will override any values entered by the user into the edit panel.

## Position (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Position receives its X, Y, Z. coordinate information either by entering it into the Position tab in the Edit Panel or by connecting an input to the Position connection on the node in the Workspace Area.

Position defines the center coordinate of the texture. By default these are offset values of the objects coordinates also sometimes referred to as the objects pivot point.

If the World Coordinates checkbox is checked in the Edit Panel then Position defines the center of the texture as offset values from the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, Z format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures position is derived from the selected Reference Objects keyframed Position values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Rotation (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Rotation receives H (Heading), P (Pitch), and B (Bank) rotational information either by entering degrees into the Rotation tab in the Edit Panel or as radians by connecting an input to the Rotation connection on the node in the Workspace Area.

Rotation defines the axial rotation around the textures center point. By default these are offset values of the objects a rotational settings.

If the World Coordinates checkbox is checked in the Edit Panel then Rotation values define the texture rotation as offset values from the world coordinates of your scene which should be fixed. You can rely on your world coordinate rotation values to always be 0,0,0, in H, P, B format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures rotation is derived from the selected Reference Objects keyframed Rotation values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Outputs

### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

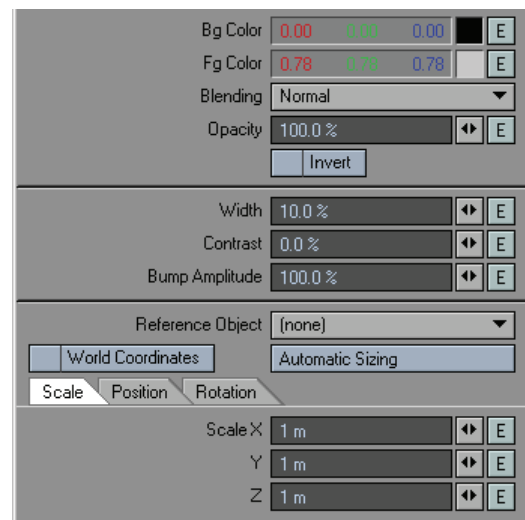
### Alpha (type: scalar):

Outputs single channel alpha (grey) information evaluated on a per spot basis.

### Bump (type: vector):

Outputs component vector direction information the length of which are the product of the bump amplitude value.

## Edit Panel

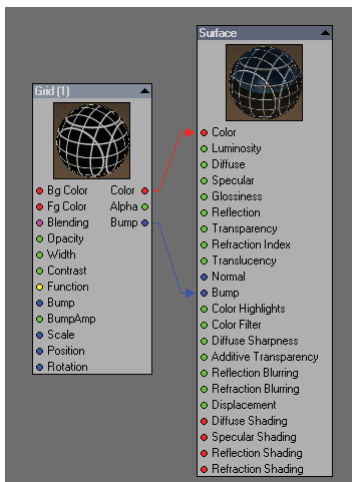
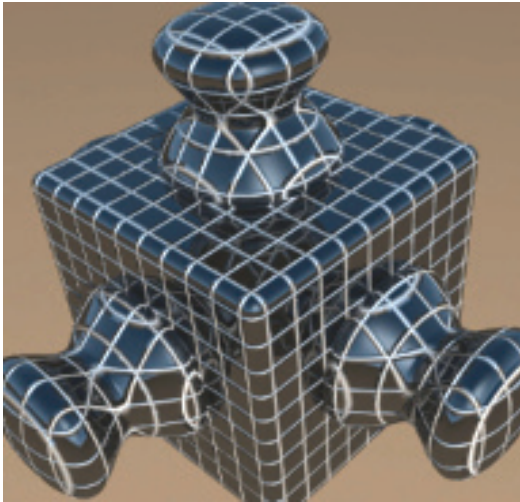


Reference Object, World Coordinates and Automatic Sizing are features exclusive to this nodes Edit Panel. Please see the explanation for these items in the chapter that covers texturing with the Surface Editor.



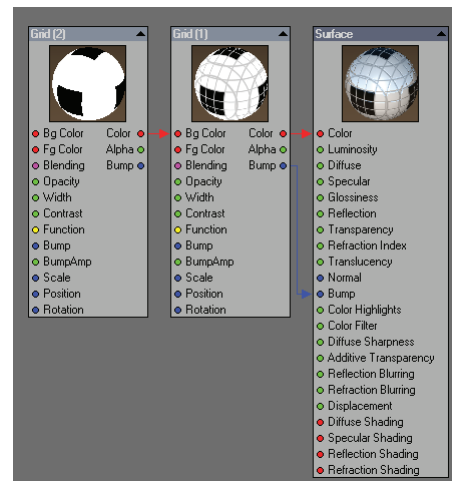
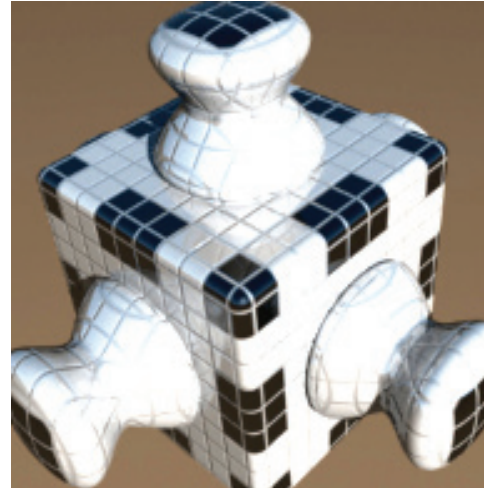
## Usage Example

Simple But Ritzy Black Bathroom tile with the Grid Node.



With 90% Contrast and 90% Width the Grid node produces a fairly convincing grout and tile texture.

Public facilities with Grid on Grid.



Any input connected to the Bg Color of the Grid node will define the color of the grid squares. This can be another Grid node as in the example above or a completely different texture node as seen in the green marble texture example in the FBM Usage Example for the FBM node description.

A mortar texture can likewise be defined on the Fg Color.





## HeteroTerrain

### Description

An experimental prototype texture developed by Dr. Forest Kenton "Ken" Musgrave, Hetero Terrain is a fractal type texture that controls the amount of details added by successive overlays (called fractalizing) according to an "Offset" value.

### Inputs

#### Bg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the background color layer.

#### Fg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the foreground color layer.

#### Blending (type: integer):

Specifies one of the blending types (0~13) found in the blending type list at the beginning of this chapter.

Can receive input from other nodes in the network or can be specified to by selecting a blending mode from the Blending pull-down menu in the Edit Panel for the node.

For general use specifying the blending mode by using the Blending pull-down menu in the Edit Panel will probably be the most desirable method of use.

For larger networks where control of the blending modes of many texture nodes simultaneously is needed or when conditional blending is desired then using a node connection to control this value may be advantageous.

#### Opacity (type: scalar):

Opacity is the amount that the selected blending mode is applied. You can think of opacity as the strength of the blend effect between the BG Color and the FG Color.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Increment (type: scalar):

This texture is a fractal or "a fractalized noise pattern", which means that it overlays successively scaled copies of the same pattern functions over previous layers in order to add detail. Technically, this is called Spectral Synthesis. These are not surface layers, but layers considered within the algorithm of the texture node itself.

Increment is the amplitude or intensity between successive levels of detail.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

### Lacunarity (type: scalar):

Lacunarity is a counterpart to the fractal dimension that describes the texture of a fractal. It has to do with the size distribution of the holes. Roughly speaking, if a fractal has large gaps or holes, it has high lacunarity; on the other hand, if a fractal is almost translationally invariant, it has low lacunarity. The lacunarity value sets the amount of change in scale between successive levels of detail.

If you're seeing repeating patterns in the texture it might be a good idea to lower the lacunarity level of the texture.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

### Octaves (type: scalar):

You can think of octaves as the number of levels of detail mentioned in the discussion just above for lacunarity. At one octave, only the basic pattern is used. At 2 octaves given a Lacunarity value of 2, a one half scale noise pattern is overlaid on top of the basic pattern. And at three octaves, the overlays are one quarter scale over the one half scale over the basic and so on.

If Lacunarity were 3 in this example, it would mean that each new layer would one third the scale of the previous layer.

Considering these three Increment, Lacunarity, and Octaves then if Lacunarity were set to 2, Increment were set to 0.5, and Octaves were 4, it would mean that each successive iteration for 4 iterations, was twice the frequency and half the amplitude.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

### Offset (type: scalar):

Defines an offset into the texture value range of the noise type where the minute details produced by the combined effects of Lacunarity, Increment, and the number of Octaves, begin.

Given a white Fg Color and a black Bg Color, an Offset of 0 will produce no details in the black areas and more details in the white areas. An Offset value of 50 will produce no details in the 50% gray areas and etc.

The Offset may vary between 0 and 100. Details are greater away from the Offset and smaller near the Offset. The Offset is relative to the noise output.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

### Function (type: function):

Connecting a function node to this input will transform the texture value of this texture based on the graph defined by the function thereby extending the range of this nodes producible patterns.

See the manual section on Function nodes for a description of how texture values can be transformed.

Can receive inputs from other function nodes in the network only.



## Bump (type: vector):

Specifies a vector direction for modifying the surface normal. This is surface normal direction information and affects the way the surface is shaded. Care should be taken when connecting to this input. Connecting dissimilar types or non directional vectors may cause the surface to shade wrongly.

Can receive input from other nodes in the network only.

## Bump Amplitude (type: scalar):

Specifies the bump height or "amplitude" of the Bump directional vectors.

Using Alpha or luminance (connecting Color) values from other textures in the network along with their respective bumps can have the effect of blending the bump textures together.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Since Bump Amplitude is a shading feature and not a displacement coordinate system value it is specified as a percentage. Specifically as a percentage of the texture value.

## Scale (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Scale receives its X, Y, Z, scale values either by entering them into the Scale tab in the Edit Panel or by connecting an input to Scale on the node in the Workspace.

Scale is the scale of the texture pattern that will be applied to your object surface.

If the World Coordinates checkbox is checked in the edit panel then Scale defines the scale of the texture in relation to the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, and Z respectively.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures scale is derived from the selected Reference Object keyframed Scale values.

Connections made in the workspace area will override any values entered by the user into the edit panel.

## Position (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Position receives its X, Y, Z, coordinate information either by entering it into the Position tab in the Edit Panel or by connecting an input to the Position connection on the node in the Workspace Area.

Position defines the center coordinate of the texture. By default these are offset values of the objects coordinates also sometimes referred to as the objects pivot point.

If the World Coordinates checkbox is checked in the Edit Panel then Position defines the center of the texture as offset values from the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, Z format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures position is derived from the selected Reference Objects keyframed Position values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Rotation (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Rotation receives H (Heading), P (Pitch), and B (Bank) rotational information either by entering degrees into the Rotation tab in the Edit Panel or as radians by connecting an input to the Rotation connection on the node in the Workspace Area.

Rotation defines the axial rotation around the textures center point. By default these are offset values of the objects a rotational settings.

If the World Coordinates checkbox is checked in the Edit Panel then Rotation values define the texture rotation as offset values from the world coordinates of your scene which should be fixed. You can rely on your world coordinate rotation values to always be 0,0,0, in H, P, B format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures rotation is derived from the selected Reference Objects keyframed Rotation values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Outputs

### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

### Alpha (type: scalar):

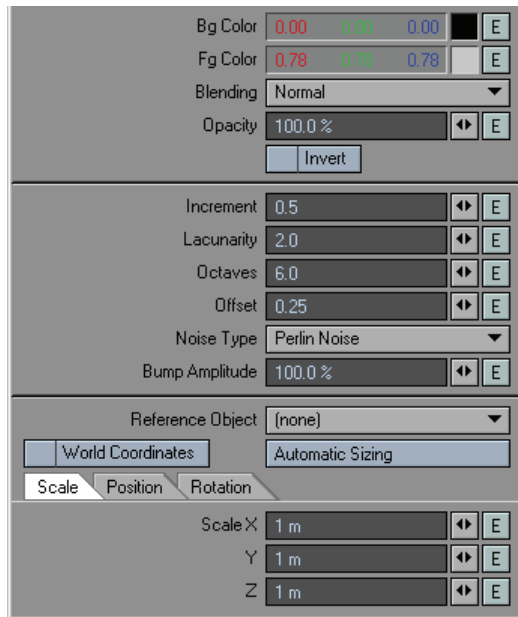
Outputs single channel alpha (grey) information evaluated on a per spot basis.

### Bump (type: vector):

Outputs component vector direction information the length of which are the product of the bump amplitude value.



## Edit Panel



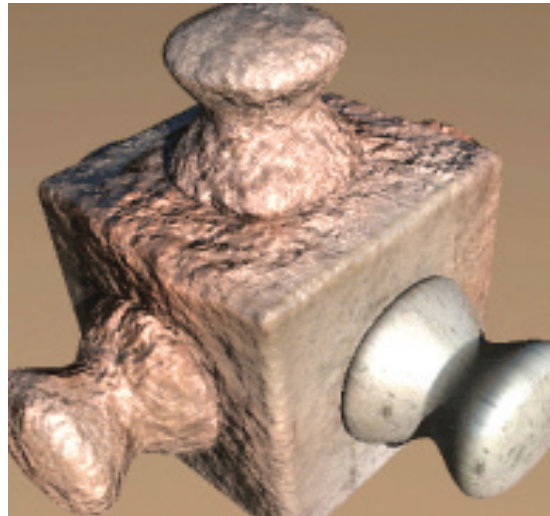
Reference Object, World Coordinates and Automatic Sizing are features exclusive to this node's Edit Panel. Please see the explanation for these items in the chapter that covers texturing with the Surface Editor.

The Noise Type selection popup menu also remains a feature exclusive to this node's Edit Panel. Without going beyond the descriptions offered in the Surface Editor section of the LightWave 3D manual for these related features, you can get a very good idea of what each Noise Type looks like by following these few simple steps:

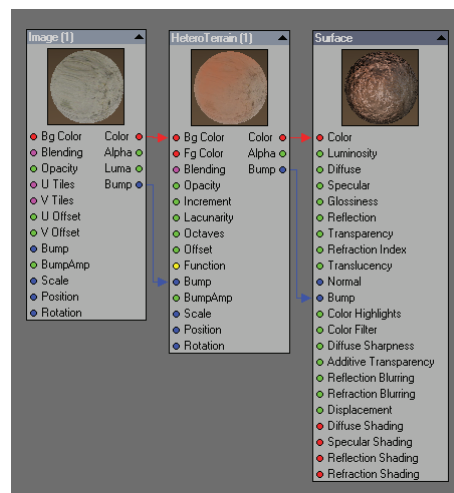
1. Set the texture Scale values small enough that you can see the full pattern of the texture on the geometry you're working with.
2. Set Increment to zero.
3. Set Lacunarity to zero.
4. Set Octaves to zero.
5. Set the Offset value anywhere between 0.25 and 0.75
6. And select the various Noise Types one at a time while watching the VIPER preview update.

## Usage Example

Hetero Terrain for extreme corrosion.



Here, the HeteroTerrain is applied over a photographic image of a scratched and marred sheet steel producing an effect that looks very much as if the object had undergone many years of weather exposed corrosion.



This example network was created very quickly by applying an image to the background color of the HeteroTerrain node and mixing the two bump maps.

Experiment on your own with this network by changing the foreground color of the HeteroTerrain texture, selecting different Noise Types, and/or replacing the loaded image with others found in the LightWave content or that you may have in your own collection.

Be sure and check out either the "3D HeteroTerrain\_Example" preset from the "Manual\_Examples" Surface Preset [F8] library or load the "3DHeteroTerrain\_CubeStubby.lwo" object that were installed with your LightWave3D content.



## Honeycomb

### Description

A procedurally generated hexagonal honeycomb pattern. The Honeycomb texture node can be applied on an axial basis X, Y, or Z.

### Inputs

#### Bg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the background (inner hexagons) color layer.

#### Fg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the foreground (lines) color layer.

#### Blending (type: integer):

Specifies one of the blending types (0~13) found in the blending type list at the beginning of this chapter.

Can receive input from other nodes in the network or can be specified to by selecting a blending mode from the Blending pull-down menu in the Edit Panel for the node.

For general use specifying the blending mode by using the Blending pull-down menu in the Edit Panel will probably be the most desirable method of use.

For larger networks where control of the blending modes of many texture nodes simultaneously is needed or when conditional blending is desired then using a node connection to control this value may be advantageous.

#### Opacity (type: scalar):

Opacity is the amount that the selected blending mode is applied. You can think of opacity as the strength of the blend effect between the BG Color and the FG Color.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Thickness (type: scalar):

Defines the thickness of the lines that form the hexagonal honeycomb pattern. This is in ratio to the size of the inner hexagons formed by the pattern. This means that increasing the thickness of the lines (Fg Color) will decrease the size of the inner hexagons (Bg Color).

Values approaching or above 0.25 will begin to distort the hexagonal shapes. Values between 1.0 and 2.0 will produce a staggered square grid pattern. A value of 2.0 or higher will cover the surface 100 percent solid in the foreground color with absolutely no pattern.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Edge Width (type: scalar):

Is the gradient falloff size of the pattern lines (foreground) into the hexagonal inner shapes. Increasing this value will have the apparent effect of decreasing the size of the inner hexagonal shapes.

The recommended value range for this parameter is 0.0 to 0.5. Higher or lower values may produce unexpected results. Values above 1.0 are not meaningful.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Function (type: function):

Connecting a function node to this input will transform the texture value of this texture based on the graph defined by the function thereby extending the range of this nodes producible patterns.

See the manual section on Function nodes for a description of how texture values can be transformed.

Can receive inputs from other function nodes in the network only.

#### Bump (type: vector):

Specifies a vector direction for modifying the surface normal. This is surface normal direction information and affects the way the surface is shaded. Care should be taken when connecting to this input. Connecting dissimilar types or non directional vectors may cause the surface to shade wrongly.

Can receive input from other nodes in the network only.



## Bump Amplitude (type: scalar):

Specifies the bump height or “amplitude” of the Bump directional vectors.

Using Alpha or luminance (connecting Color) values from other textures in the network along with their respective bumps can have the effect of blending the bump textures together.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Since Bump Amplitude is a shading feature and not a displacement coordinate system value it is specified as a percentage. Specifically as a percentage of the texture value.

## Scale (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Scale receives its X, Y, Z, scale values either by entering them into the Scale tab in the Edit Panel or by connecting an input to Scale on the node in the Workspace.

Scale is the scale of the texture pattern that will be applied to your object surface.

If the World Coordinates checkbox is checked in the edit panel then Scale defines the scale of the texture in relation to the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, and Z respectively.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures scale is derived from the selected Reference Object keyframed Scale values.

Connections made in the workspace area will override any values entered by the user into the edit panel.

## Position (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Position receives its X, Y, Z, coordinate information either by entering it into the Position tab in the Edit Panel or by connecting an input to the Position connection on the node in the Workspace Area.

Position defines the center coordinate of the texture. By default these are offset values of the objects coordinates also sometimes referred to as the objects pivot point.

If the World Coordinates checkbox is checked in the Edit Panel then Position defines the center of the texture as offset values

from the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, Z format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures position is derived from the selected Reference Objects keyframed Position values.

## Rotation (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Rotation receives H (Heading), P (Pitch), and B (Bank) rotational information either by entering degrees into the Rotation tab in the Edit Panel or as radians by connecting an input to the Rotation connection on the node in the Workspace Area.

Rotation defines the axial rotation around the textures center point. By default these are offset values of the objects a rotational settings.

If the World Coordinates checkbox is checked in the Edit Panel then Rotation values define the texture rotation as offset values from the world coordinates of your scene which should be fixed. You can rely on your world coordinate rotation values to always be 0,0,0, in H, P, B format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures rotation is derived from the selected Reference Objects keyframed Rotation values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Outputs

### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

### Alpha (type: scalar):

Outputs single channel alpha (grey) information evaluated on a per spot basis.

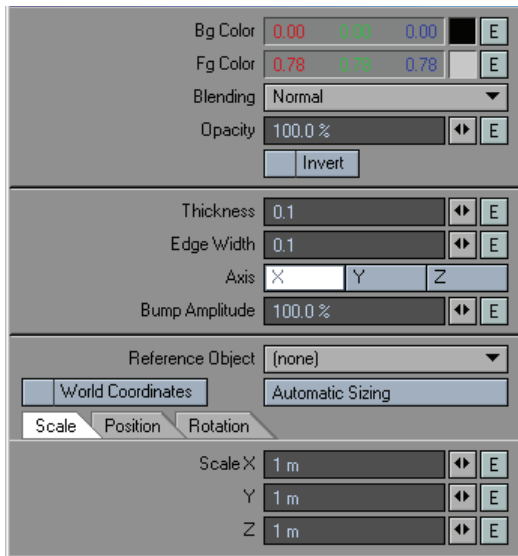
### Bump (type: vector):

Outputs component vector direction information the length of which are the product of the bump amplitude value.





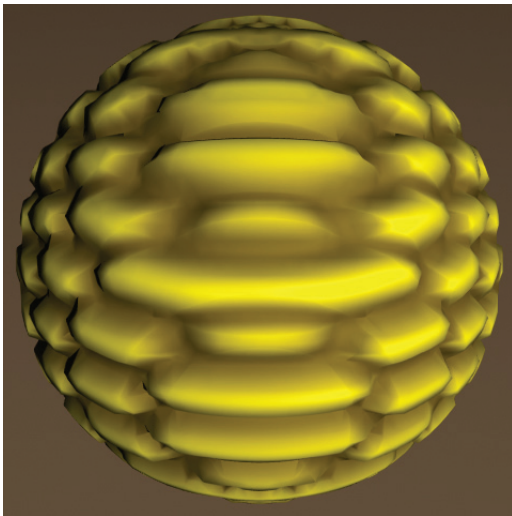
## Edit Panel



Axis are not offered as node connections in the Workspace area. Likewise Reference Object, World Coordinates and Automatic Sizing are features exclusive to this node's Edit Panel. Please see the explanation for these items in the chapter that covers texturing with the Surface Editor.

## Usage Example

Funky bug body bumps with displaced honeycomb.



This example was created by applying the Honeycomb Color output to the Displacement input of the Surface destination node and also by applying the Color output of a copy of the Honeycomb node once set up, to the Color input of the Surface destination node.

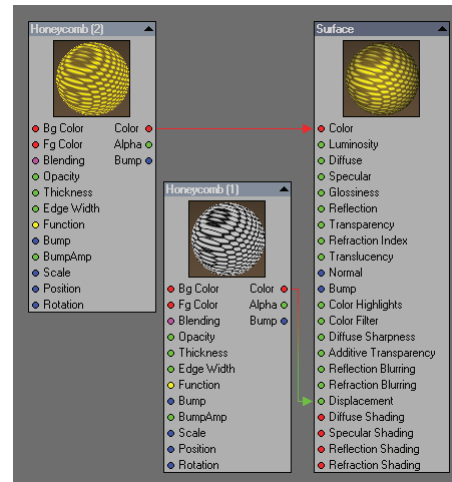
The Color output was used on the Displacement input for a specific reason. When connected in this way you gain control over how much the foreground and background colors affect the displacement amount within the limits of the Distance value set in the Object Properties panel.

It's important to note that to use the Displacement input on the Surface destination node Enable Bump in the Deform tab of the Object Properties panel must be checked and a Distance value greater than zero must be entered.

For this example the Enable Bump box was checked and 2cm was entered for the distance value.

A 5 Segment Tessellation sphere was used for the object.

Here we can see the simple node network that produces this effect.



Of particular interest in this network is the Color connection. When colors are used in the same basic way that the texture values are used for displacement the results are very similar to ambient occlusion but without the increased rendering time or render pass management that may be associated with real ambient occlusion.

This occlusive effect and its advantages may be extended by applying this technique to other channels besides Color. For example Diffuse, Reflection, and Specularity.





## Hybrid-MultiFractal

### Description

An experimental fractal prototype texture developed by Dr. Forest Kenton “Ken” Musgrave, Hybrid Multifractal attempts to control the amount of details according to the slope of the underlying overlays. Hybrid Multifractal is conventionally used to generate terrains with smooth valley areas and rough peaked mountains. With high Lacunarity values, it tends to produce embedded plateaus.

### Inputs

#### Bg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the background color layer.

#### Fg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the foreground color layer.

#### Blending (type: integer):

Specifies one of the blending types (0~13) found in the blending type list at the beginning of this chapter.

Can receive input from other nodes in the network or can be specified to by selecting a blending mode from the Blending pull-down menu in the Edit Panel for the node.

For general use specifying the blending mode by using the Blending pull-down menu in the Edit Panel will probably be the most desirable method of use. However, for larger networks where control of the blending modes of many texture nodes simultaneously is needed or when conditional blending is desired then using a node connection to control this value may be advantageous.

#### Opacity (type: scalar):

Opacity is the amount that the selected blending mode is applied. You can think of opacity as the strength of the blend effect between the BG Color and the FG Color.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Increment (type: scalar):

Since this texture is a fractal or a fractalized noise pattern, it means it can overlay successive scaled copies of the same pattern functions over previous layers in order to add detail. Technically, this is called Spectral Synthesis. These are not surface layers, but layers considered within the algorithm of the texture node itself.

Increment is the amplitude or intensity between successive levels of detail.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Lacunarity (type: scalar):

Lacunarity is a counterpart to the fractal dimension that describes the texture of a fractal. It has to do with the size distribution of the holes.

Roughly speaking, if a fractal has large gaps or holes, it has high lacunarity; on the other hand, if a fractal is almost translationally invariant, it has low lacunarity. The lacunarity value sets the amount of change in scale between successive levels of detail.

If you're seeing repeating patterns in the texture it might be a good idea to lower the lacunarity level of the texture.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Octaves (type: scalar):

You can think of octaves as the number of levels of detail mentioned in the discussion just above for lacunarity. At one octave, only the basic pattern is used. At 2 octaves given a Lacunarity value of 2, a one half scale noise pattern is overlaid on top of the basic pattern. And at three octaves, the overlays are one quarter scale over the one half scale over the basic and so on.

If Lacunarity were 3 in this example, it would mean that each new layer would one third the scale of the previous layer.

Considering these three Increment, Lacunarity, and Octaves then if Lacunarity were set to 2, Increment were set to 0.5, and Octaves were 4, it would mean that each successive iteration for 4 iterations, was twice the frequency and half the amplitude.

Can receive patterns or numerical inputs from other nodes. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Offset (type: scalar):

Defines an offset into the texture value range where the minute details produced by the combined effects of Lacunarity, Increment, and the number of Octaves, begin to ramp sharply.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.



## Function (type: function):

Connecting a function node to this input will transform the texture value of this texture based on the graph defined by the function thereby extending the range of this nodes producible patterns.

See the manual section on Function nodes for a description of how texture values can be transformed.

Can receive inputs from other function nodes in the network only.

## Bump (type: vector):

Specifies a vector direction for modifying the surface normal. This is surface normal direction information and affects the way the surface is shaded. Care should be taken when connecting to this input. Connecting dissimilar types or non directional vectors may cause the surface to shade wrongly.

Can receive input from other nodes in the network only.

## Bump Amplitude (type: scalar):

Specifies the bump height or "amplitude" of the Bump directional vectors.

Using Alpha or luminance (connecting Color) values from other textures in the network along with their respective bumps can have the effect of blending the bump textures together.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Since Bump Amplitude is a shading feature and not a displacement coordinate system value it is specified as a percentage. Specifically as a percentage of the texture value.

## Scale (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Scale receives its X, Y, Z, scale values either by entering them into the Scale tab in the Edit Panel or by connecting an input to Scale on the node in the Workspace.

Scale is the scale of the texture pattern that will be applied to your object surface.

If the World Coordinates checkbox is checked in the edit panel then Scale defines the scale of the texture in relation to the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, and Z respectively.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures scale is derived from the selected Reference Object keyframed Scale values.

Connections made in the workspace area will override any values entered by the user into the edit panel.

## Position (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Position receives its X, Y, Z. coordinate information either by entering it into the Position tab in the Edit Panel or by connecting an input to the Position connection on the node in the Workspace Area.

Position defines the center coordinate of the texture. By default these are offset values of the objects coordinates also sometimes referred to as the objects pivot point.

If the World Coordinates checkbox is checked in the Edit Panel then Position defines the center of the texture as offset values from the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, Z format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures position is derived from the selected Reference Objects keyframed Position values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Rotation (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Rotation receives H (Heading), P (Pitch), and B (Bank) rotational information either by entering degrees into the Rotation tab in the Edit Panel or as radians by connecting an input to the Rotation connection on the node in the Workspace Area.

Rotation defines the axial rotation around the textures center point. By default these are offset values of the objects a rotational settings.

If the World Coordinates checkbox is checked in the Edit Panel then Rotation values define the texture rotation as offset values from the world coordinates of your scene which should be fixed. You can rely on your world coordinate rotation values to always be 0,0,0, in H, P, B format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures rotation is derived from the selected Reference Objects keyframed Rotation values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.



## Outputs

### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

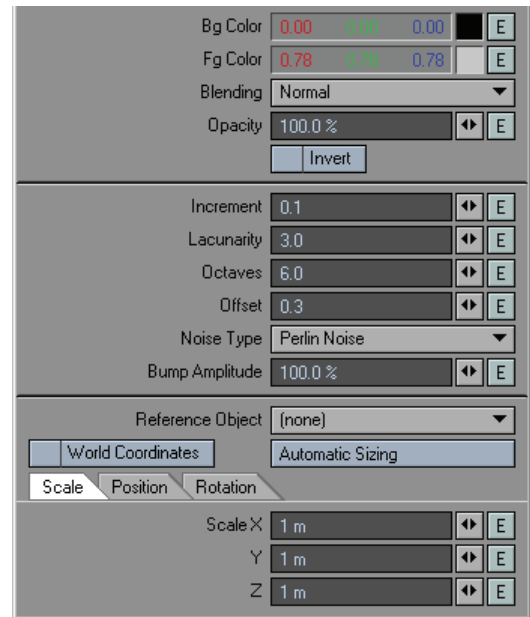
### Alpha (type: scalar):

Outputs single channel alpha (grey) information evaluated on a per spot basis.

### Bump (type: vector):

Outputs component vector direction information the length of which are the product of the bump amplitude value.

## Edit Panel



Reference Object, World Coordinates and Automatic Sizing are features exclusive to this nodes Edit Panel. Please see the explanation for these items in the chapter that covers texturing with the Surface Editor.

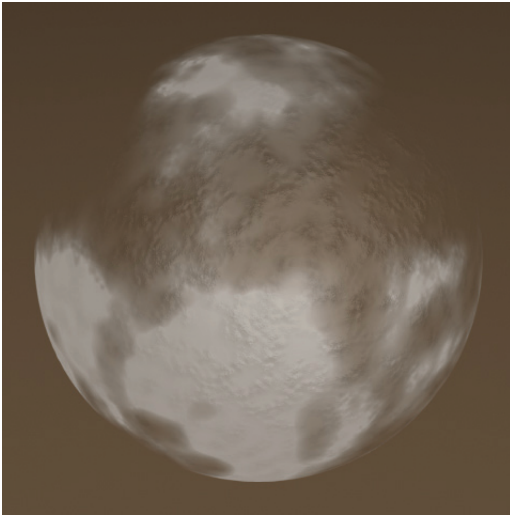
The Noise Type selection popup menu also remains a feature exclusive to this nodes Edit Panel. Without going beyond the descriptions offered in the Surface Editor section of this manual for these related features, you can get a very good idea of what each Noise Type looks like by following these few simple steps:

1. Set the texture Scale values small enough that you can see the full pattern of the texture on the geometry you're working with.
2. Set Increment to zero.
3. Set Lacunarity to zero.
4. Set Octaves to zero.
5. Set the Offset value anywhere between 0.25 and 0.75
6. And select the various Noise Types one at a time while watching the VIPER preview update.



## Usage Example

Ice and slush with Hybrid-MultiFractal.

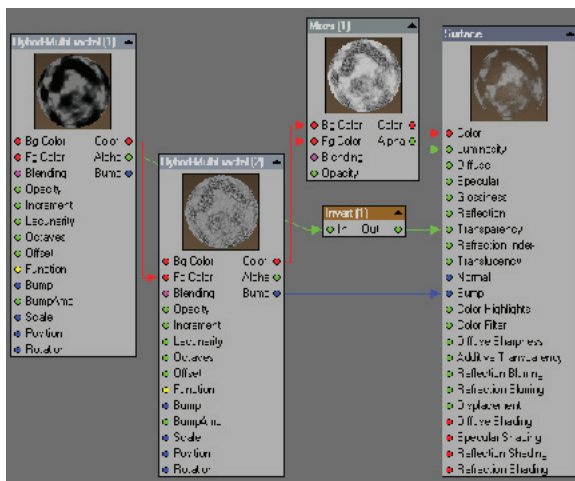


Here Hybrid-MultiFractal is being used to produce a melting ice covered lake surface.

Actually two Hybrid-MultiFractal nodes are being used with very slight alterations to the Lacunarity and with different noise types.

Hybrid-MultiFractal (1) with a Scale setting of 30cm on X, Y, and Z (on a 1m sphere) is being used to define the transparent or slushy areas of the frozen surface from the opaque or thicker solid ice areas.

Hybrid-MultiFractal (2) with a scale value of 4cm for X, Y, and Z, defines the windblown granular surface attributes commonly associated with naturally occurring ice and slush.



Be sure and check out the "3D HybridMultiFractal\_Example" preset from the "Manual\_Examples" Surface Preset [F8] library or load the "3DHybrid-MultiFractal\_Ball.lwo" object that were installed with your LightWave3D content.

## Marble

### Description

A texture that produces a three dimensional grid of columns of marbly vein-like patterns. The Marble texture can be applied on an axial basis X, Y, or Z.

### Inputs

#### Bg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the background color layer. The BgColor is the color between the veins.

#### Fg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the foreground color layer. The Fg Color is the color of the veins.

#### Blending (type: integer):

Specifies one of the blending types (0~13) found in the blending type list at the beginning of this chapter.

Can receive input from other nodes in the network or can be specified to by selecting a blending mode from the Blending pull-down menu in the Edit Panel for the node.

For general use specifying the blending mode by using the Blending pull-down menu in the Edit Panel will probably be the most desirable method of use.

For larger networks where control of the blending modes of many texture nodes simultaneously is needed or when conditional blending is desired then using a node connection to control this value may be advantageous.

#### Opacity (type: scalar):

Opacity is the amount that the selected blending mode is applied. You can think of opacity as the strength of the blend effect between the BG Color and the FG Color.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.



## Vein Spacing (type: scalar):

Defines the number and size of the veins within the given Scale value.

For example, given a uniform scale of 1m a Vein Spacing value of 1.0 would produce one vein per meter. A vein spacing value of 0.5 would produce two veins per meter, 0.25 four veins and so on.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

## Distortion (type: scalar):

Controls the amount of distortion or noise that is applied to the veins. This property is in direct correlation with Noise Scale.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

## Noise Scale (type: scalar):

Controls the scale of the noise that is applied to the vein columns. This property is in direct correlation with the Distortion value.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify values for this attribute using the controls available in the Edit Panel for this node.

## Contrast (type: scalar):

Sets the contrast of the texture value gradient. Higher values produce less gradient information between texture value extremes and therefore less noise is produced. The lower values spread the gradient more evenly between the extremes of the texture values and thus allow for more overall noise in the final output from this node

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify the values for this attribute using the controls available in the Edit Panel for this node.

## Frequencies (type: scalar):

"Frequencies" as used in this node is really just another name for octaves as defined in some of the descriptions of the fractal type textures within this document.

You can think of "Frequencies" as the number of levels of detail. At one "Frequency", only the basic pattern is used.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify the values for this attribute using the controls available in the Edit Panel for this node.

## Function (type: function):

Connecting a function node to this input will transform the texture value of this texture based on the graph defined by the function thereby extending the range of this nodes producible patterns.

See the manual section on Function nodes for a description of how texture values can be transformed.

Can receive inputs from other function nodes in the network only.

## Bump (type: vector):

Specifies a vector direction for modifying the surface normal. This is surface normal direction information and affects the way the surface is shaded. Care should be taken when connecting to this input. Connecting dissimilar types or non directional vectors may cause the surface to shade wrongly.

Can receive input from other nodes in the network only.

## Bump Amplitude (type: scalar):

Specifies the bump height or "amplitude" of the Bump directional vectors.

Using Alpha or luminance (connecting Color) values from other textures in the network along with their respective bumps can have the effect of blending the bump textures together.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Since Bump Amplitude is a shading feature and not a displacement coordinate system value it is specified as a percentage. Specifically as a percentage of the texture value.

## Scale (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Scale receives its X, Y, Z, scale values either by entering them into the Scale tab in the Edit Panel or by connecting an input to Scale on the node in the Workspace.

Scale is the scale of the texture pattern that will be applied to your object surface.

If the World Coordinates checkbox is checked in the edit panel then Scale defines the scale of the texture in relation to the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, and Z respectively.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures scale is derived from the selected Reference Object keyframed Scale values.

Connections made in the workspace area will override any values entered by the user into the edit panel.



## Position (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Position receives its X, Y, Z. coordinate information either by entering it into the Position tab in the Edit Panel or by connecting an input to the Position connection on the node in the Workspace Area.

Position defines the center coordinate of the texture. By default these are offset values of the objects coordinates also sometimes referred to as the objects pivot point.

If the World Coordinates checkbox is checked in the Edit Panel then Position defines the center of the texture as offset values from the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, Z format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures position is derived from the selected Reference Objects keyframed Position values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Rotation (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Rotation receives H (Heading), P (Pitch), and B (Bank) rotational information either by entering degrees into the Rotation tab in the Edit Panel or as radians by connecting an input to the Rotation connection on the node in the Workspace Area.

Rotation defines the axial rotation around the textures center point. By default these are offset values of the objects a rotational settings.

If the World Coordinates checkbox is checked in the Edit Panel then Rotation values define the texture rotation as offset values from the world coordinates of your scene which should be fixed. You can rely on your world coordinate rotation values to always be 0,0,0, in H, P, B format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures rotation is derived from the selected Reference Objects keyframed Rotation values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Outputs

### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

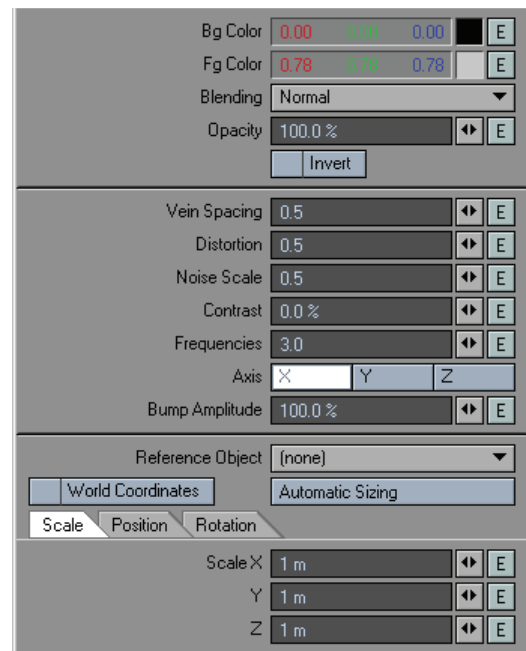
### Alpha (type: scalar):

Outputs single channel alpha (grey) information evaluated on a per spot basis.

### Bump (type: vector):

Outputs component vector direction information the length of which are the product of the bump amplitude value.

## Edit Panel



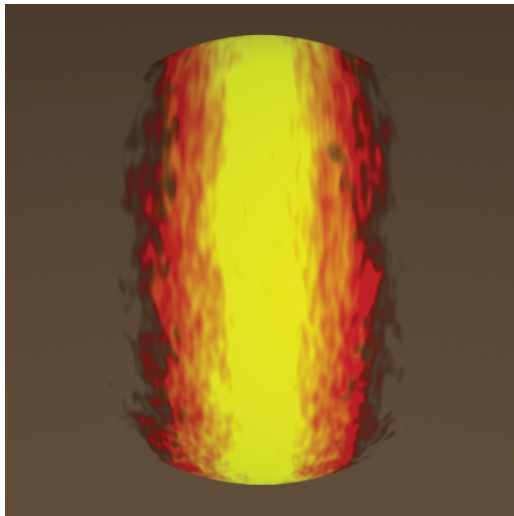
Axis are not offered as node connections in the Workspace area. Likewise Reference Object, World Coordinates and Automatic Sizing are features exclusive to this nodes Edit Panel. Please see the explanation for these items in the chapter that covers texturing with the Surface Editor.





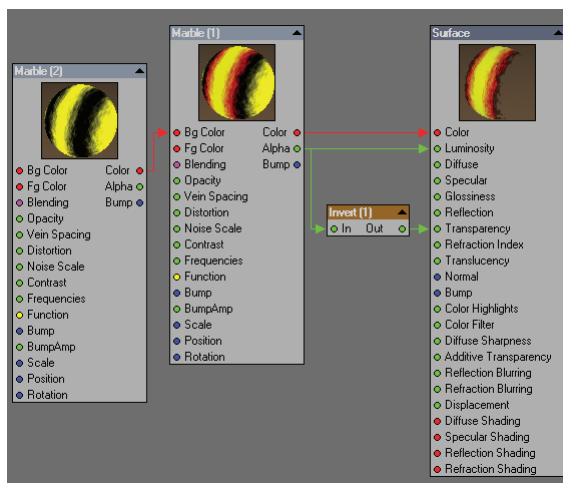
## Usage Example

The painted ring of fire - a non-marble marble excursion - Animated.



This example takes advantage of the vein and noise properties of the Marble texture node in order to produce a column of fire-like surface coloring.

Here is the node network that produces the result.



A second copy of the Marble texture node is being used to define the transparent areas and additionally feather out the edges of the flames from opaque to transparent areas.

This example is animated so selecting Make Preview in the Preview pull-down menu on the VIPER panel will allow you to create and playback an animation based on the values specified in the node network.



## MultiFractal

### Description

A multipurpose fractal type texture node.

### Inputs

#### Bg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the background color layer.

#### Fg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the foreground color layer.

#### Blending (type: integer):

Specifies one of the blending types (0~13) found in the blending type list at the beginning of this chapter.

Can receive input from other nodes in the network or can be specified to by selecting a blending mode from the Blending pull-down menu in the Edit Panel for the node.

For general use specifying the blending mode by using the Blending pull-down menu in the Edit Panel will probably be the most desirable method of use. However, for larger networks where control of the blending modes of many texture nodes simultaneously is needed or when conditional blending is desired then using a node connection to control this value may be advantageous.

#### Opacity (type: scalar):

Opacity is the amount that the selected blending mode is applied. You can think of opacity as the strength of the blend effect between the BG Color and the FG Color.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Increment (type: scalar):

Since this texture is a fractal or a fractalized noise pattern, it means that we can overlay smaller and smaller copies of the same pattern functions over previous layers in order to add detail. Technically, this is called Spectral Synthesis. These are not surface layers, but layers considered within the algorithm of the texture node itself.

Increment is the amplitude or intensity between successive levels of detail.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Lacunarity (type: scalar):

Lacunarity is a counterpart to the fractal dimension that describes the texture of a fractal. It has to do with the size distribution of the holes. Roughly speaking, if a fractal has large gaps or holes, it has high lacunarity; on the other hand, if a fractal is almost translationally invariant, it has low lacunarity. The lacunarity value sets the amount of change in scale between successive levels of detail.

If you're seeing repeating patterns in the texture it might be a good idea to lower the lacunarity level of the texture.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Octaves (type: scalar):

You can think of octaves as the number of levels of detail mentioned in the discussion just above for lacunarity. At one octave, only the basic pattern is used. At 2 octaves given a Lacunarity value of 2, a one half scale noise pattern is overlaid on top of the basic pattern. And at three octaves, the overlays are one quarter scale over the one half scale over the basic and so on.

If Lacunarity were 3 in this example, it would mean that each new layer would one third the scale of the previous layer.

Considering these three Increment, Lacunarity, and Octaves then if Lacunarity were set to 2, Increment were set to 0.5, and Octaves were 4, it would mean that each successive iteration for 4 iterations, was twice the frequency and half the amplitude.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Offset (type: scalar):

Defines an offset into the texture value range where the minute details produced by the combined effects of Lacunarity, Increment, and the number of Octaves, begin to ramp sharply.



## Function (type: function):

Connecting a function node to this input will transform the texture value of this texture based on the graph defined by the function thereby extending the range of this nodes producible patterns.

See the manual section on Function nodes for a description of how texture values can be transformed.

Can receive inputs from other function nodes in the network only.

## Bump (type: vector):

Specifies a vector direction for modifying the surface normal. This is surface normal direction information and affects the way the surface is shaded. Care should be taken when connecting to this input. Connecting dissimilar types or non directional vectors may cause the surface to shade wrongly.

Can receive input from other nodes in the network only.

## Bump Amplitude (type: scalar):

Specifies the bump height or "amplitude" of the Bump directional vectors.

Using Alpha or luminance (connecting Color) values from other textures in the network along with their respective bumps can have the effect of blending the bump textures together.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Since Bump Amplitude is a shading feature and not a displacement coordinate system value it is specified as a percentage. Specifically as a percentage of the texture value.

## Scale (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Scale receives its X, Y, Z, scale values either by entering them into the Scale tab in the Edit Panel or by connecting an input to Scale on the node in the Workspace.

Scale is the scale of the texture pattern that will be applied to your object surface.

If the World Coordinates checkbox is checked in the edit panel then Scale defines the scale of the texture in relation to the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, and Z respectively.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures scale is derived from the selected Reference Object keyframed Scale values.

Connections made in the workspace area will override any values entered by the user into the edit panel.

## Position (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Position receives its X, Y, Z, coordinate information either by entering it into the Position tab in the Edit Panel or by connecting an input to the Position connection on the node in the Workspace Area.

Position defines the center coordinate of the texture. By default these are offset values of the objects coordinates also sometimes referred to as the objects pivot point.

If the World Coordinates checkbox is checked in the Edit Panel then Position defines the center of the texture as offset values from the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, Z format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures position is derived from the selected Reference Objects keyframed Position values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Rotation (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Rotation receives H (Heading), P (Pitch), and B (Bank) rotational information either by entering degrees into the Rotation tab in the Edit Panel or as radians by connecting an input to the Rotation connection on the node in the Workspace Area.

Rotation defines the axial rotation around the textures center point. By default these are offset values of the objects a rotational settings.

If the World Coordinates checkbox is checked in the Edit Panel then Rotation values define the texture rotation as offset values from the world coordinates of your scene which should be fixed. You can rely on your world coordinate rotation values to always be 0,0,0, in H, P, B format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures rotation is derived from the selected Reference Objects keyframed Rotation values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.



## Outputs

### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

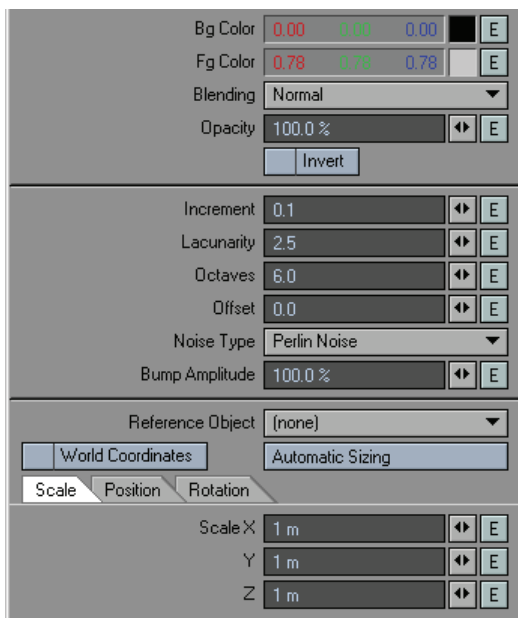
### Alpha (type: scalar):

Outputs single channel alpha (grey) information evaluated on a per spot basis.

### Bump (type: vector):

Outputs component vector direction information the length of which are the product of the bump amplitude value.

## Edit Panel



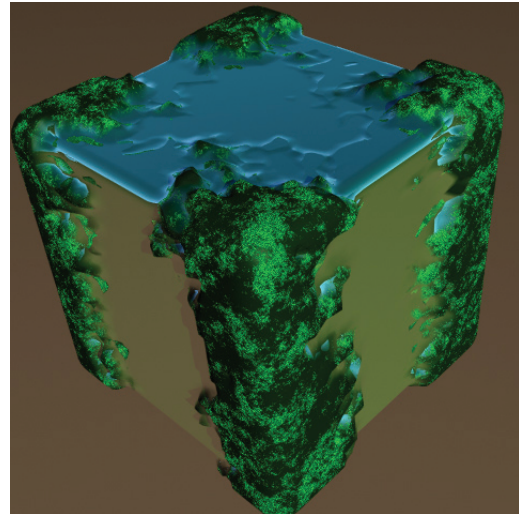
Reference Object, World Coordinates and Automatic Sizing are features exclusive to this nodes Edit Panel. Please see the explanation for these items in the chapter that covers texturing with the Surface Editor.

The Noise Type selection popup menu also remains a feature exclusive to this nodes Edit Panel. Without going beyond the descriptions offered in the Surface Editor section of this manual for these related features, you can get a very good idea of what each Noise Type looks like by following these few simple steps:

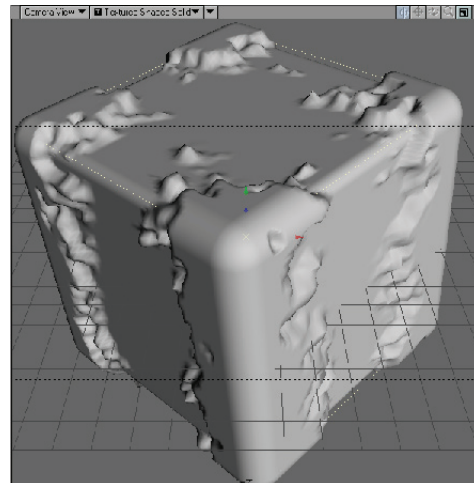
1. Set the texture Scale values small enough that you can see the full pattern of the texture on the geometry you're working with.
2. Set Increment to zero.
3. Set Lacunarity to zero.
4. Set Octaves to zero.
5. Set the Offset value anywhere between 0.25 and 0.75
6. And select the various Noise Types one at a time while watching the VIPER preview update.

## Usage Example

Alien moss and slime growth with displaced MultiFractal, Animated.



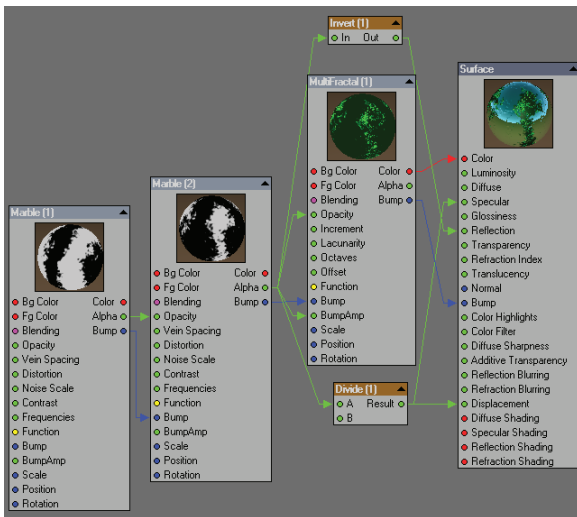
Example rendering using green colors.



Here's a screen shot of the cube object after displacement.

This example looks much more complex than it actually is. The vein column aspect of the Marble texture is being used for the Displacement of the subpatched object and as the Opacity for the MultiFractal texture. Used in this way the MultiFractal texture is placed only in those same areas that the displacement occurs - wherever the Marble is affecting.

The Marble texture as we discussed in its description, produces a grid of column shaped veins. For this example those columns were spaced and placed so that they would pass through the four corners of the cube as you see in the adjacent rendered image.



It's important to note that in order to use the Displacement input on the Surface destination node Enable Bump in the Deform tab of the Object Properties panel must be checked and a Distance value greater than zero must be entered.

For this example the Enable Bump box was checked and 20cm was entered for the Distance value. A value of 32 was used the Renderer SubPatch Per Object Level.

Be sure and check out the "3D MultiFractal\_Example" preset from the "Manual\_Examples" Surface Preset [F8] library or load the "3DMultiFractal\_Cube.lwo" object, that were installed with your LightWave3D content.

This example is animated and the animation affects both the textures and the displacements. To create the animation select a first and last frame for the animation length, a file format to render to and press [F10] to render the animation.

## RidgedMultiFractal

### Description

An experimental fractal prototype texture developed by Dr. Forest Kenton "Ken" Musgrave, Ridged MultiFractal is one of the best Fractal texture patterns to produce mountains, and mountainous terrain. It introduces highly detailed peaks and ridges often in similar shapes as naturally occurring mountain ranges. Control over the sharpness of the ridges can be achieved by adjusting the Threshold value. The Noise Types used are fairly critical in determining how mountainous the terrain should or should not be.

### Inputs

#### Bg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the background color layer.

#### Fg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the foreground color layer.

#### Blending (type: integer):

Specifies one of the blending types (0~13) found in the blending type list at the beginning of this chapter.

Can receive input from other nodes in the network or can be specified to by selecting a blending mode from the Blending pull-down menu in the Edit Panel for the node.

For general use specifying the blending mode by using the Blending pull-down menu in the Edit Panel will probably be the most desirable method of use. However, for larger networks where control of the blending modes of many texture nodes simultaneously is needed or when conditional blending is desired then using a node connection to control this value may be advantageous.

#### Opacity (type: scalar):

Opacity is the amount that the selected blending mode is applied. You can think of opacity as the strength of the blend effect between the BG Color and the FG Color.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.



## Increment (type: scalar):

Since this texture is a fractal or a fractalized noise pattern, it means that it overlays successively scaled copies of the same pattern functions over previous layers in order to add detail. Technically, this is called Spectral Synthesis. These are not surface layers, but layers considered within the algorithm of the texture node itself.

Increment is the amplitude or intensity between successive levels of detail.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

## Lacunarity (type: scalar):

Lacunarity is a counterpart to the fractal dimension that describes the texture of a fractal. It has to do with the size distribution of the holes.

Roughly speaking, if a fractal has large gaps or holes, it has high lacunarity; on the other hand, if a fractal is almost translationally invariant, it has low lacunarity. The lacunarity value sets the amount of change in scale between successive levels of detail.

If you're seeing repeating patterns in the texture it might be a good idea to lower the lacunarity level of the texture.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

## Octaves (type: scalar):

You can think of octaves as the number of levels of detail mentioned in the discussion just above for lacunarity. At one octave, only the basic pattern is used. At 2 octaves given a Lacunarity value of 2, a one half scale noise pattern is overlaid on top of the basic pattern. And at three octaves, the overlays are one quarter scale over the one half scale over the basic and so on.

If Lacunarity were 3 in this example, it would mean that each new layer would one third the scale of the previous layer.

Considering these three Increment, Lacunarity, and Octaves then if Lacunarity were set to 2, Increment were set to 0.5, and Octaves were 4, it would mean that each successive iteration for 4 iterations, was twice the frequency and half the amplitude.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

## Offset (type: scalar):

Defines an offset into the texture value range where the minute details produced by the combined effects of Lacunarity, Increment, and the number of Octaves, begin to ramp sharply.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

## Threshold (type: scalar):

The threshold value controls a threshold level into the texture value range where the results of the combined Increment, Lacunarity, and Octaves will affect.

Higher Threshold values will create more texture detail beginning in the areas of higher texture value.

So for example, where a value of 2 would add detail (sharp peaks) to the higher areas with almost no detail in the lower areas, a value of 10 would place extreme detail in the higher areas and considerable detail lower down. A value of 100 would flush the entire texture with extreme detail. Of course relevant to the other parameters used to define the texture.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

## Function (type: function):

Connecting a function node to this input will transform the texture value of this texture based on the graph defined by the function thereby extending the range of this nodes producible patterns.

See the manual section on Function nodes for a description of how texture values can be transformed.

Can receive inputs from other function nodes in the network only.

## Bump (type: vector):

Specifies a vector direction for modifying the surface normal. This is surface normal direction information and affects the way the surface is shaded. Care should be taken when connecting to this input. Connecting dissimilar types or non directional vectors may cause the surface to shade wrongly.

Can receive input from other nodes in the network only.

## Bump Amplitude (type: scalar):

Specifies the bump height or "amplitude" of the Bump directional vectors.

Using Alpha or luminance (connecting Color) values from other textures in the network along with their respective bumps can have the effect of blending the bump textures together.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Since Bump Amplitude is a shading feature and not a displacement coordinate system value it is specified as a percentage. Specifically as a percentage of the texture value.

## Scale (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Scale receives its X, Y, Z, scale values either by entering them into the Scale tab in the Edit Panel or by connecting an input to Scale on the node in the Workspace.





Scale is the scale of the texture pattern that will be applied to your object surface.

If the World Coordinates checkbox is checked in the edit panel then Scale defines the scale of the texture in relation to the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, and Z respectively.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures scale is derived from the selected Reference Object keyframed Scale values.

Connections made in the workspace area will override any values entered by the user into the edit panel.

## Position (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Position receives its X, Y, Z. coordinate information either by entering it into the Position tab in the Edit Panel or by connecting an input to the Position connection on the node in the Workspace Area.

Position defines the center coordinate of the texture. By default these are offset values of the objects coordinates also sometimes referred to as the objects pivot point.

If the World Coordinates checkbox is checked in the Edit Panel then Position defines the center of the texture as offset values from the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, Z format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures position is derived from the selected Reference Objects keyframed Position values.

## Rotation (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Rotation receives H (Heading), P (Pitch), and B (Bank) rotational information either by entering degrees into the Rotation tab in the Edit Panel or as radians by connecting an input to the Rotation connection on the node in the Workspace Area.

Rotation defines the axial rotation around the textures center point. By default these are offset values of the objects a rotational settings.

If the World Coordinates checkbox is checked in the Edit Panel then Rotation values define the texture rotation as offset values from the world coordinates of your scene which should be fixed. You can rely on your world coordinate rotation values to always be 0,0,0, in H, P, B format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures rotation is derived from the selected Reference Objects keyframed Rotation values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Outputs

### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

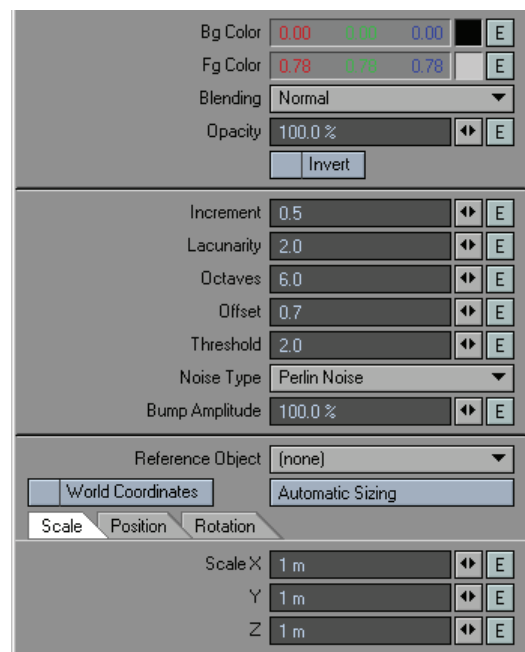
### Alpha (type: scalar):

Outputs single channel alpha (grey) information evaluated on a per spot basis.

### Bump (type: vector):

Outputs component vector direction information the length of which are the product of the bump amplitude value.

## Edit Panel



Reference Object, World Coordinates and Automatic Sizing are features exclusive to this nodes Edit Panel. Please see the explanation for these items in the chapter that covers texturing with the Surface Editor.

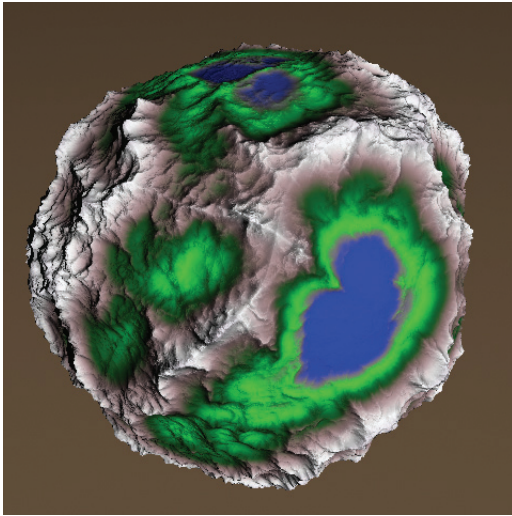
The Noise Type selection popup menu also remains a feature exclusive to this nodes Edit Panel. Without going beyond the descriptions offered in the Surface Editor section of this manual for these related features, you can get a very good idea of what each Noise Type looks like by following these few simple steps:

1. Set the texture Scale values small enough that you can see the full pattern of the texture on the geometry you're working with.
2. Set Increment to zero.
3. Set Lacunarity to zero.
4. Set Octaves to zero.
5. Set the Offset value anywhere between 0.25 and 0.75
6. And select the various Noise Types one at a time while watching the VIPER preview update.

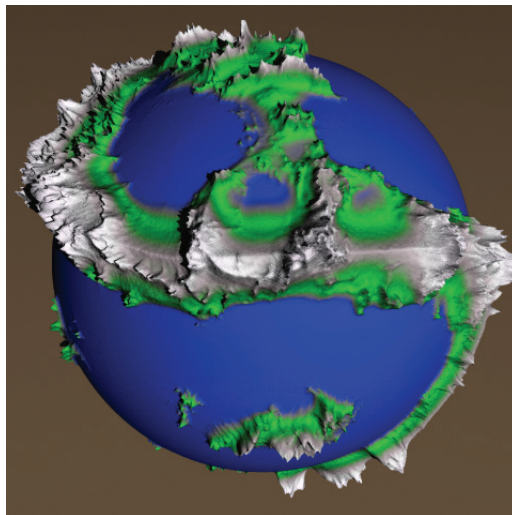


## Usage Example

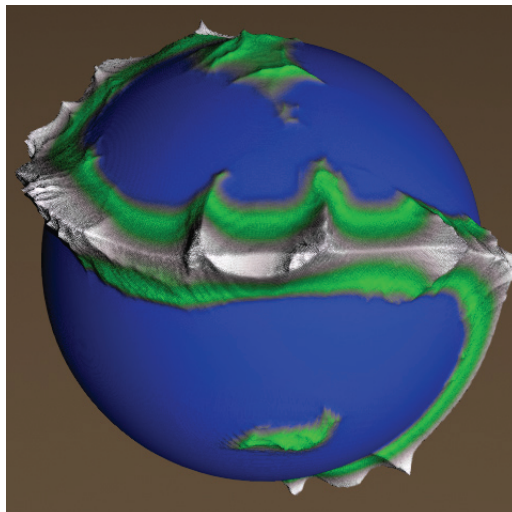
Ridged MultiFractal - The ultimate landscape generator.



Ridged MultiFractal example using Perlin Noise and a Threshold value of 2.

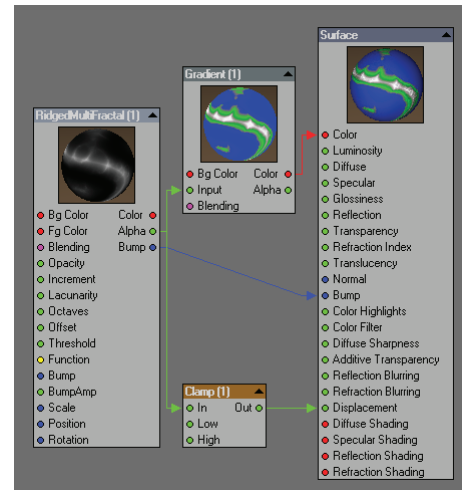


Ridged MultiFractal example using Lattice Convolution Noise and a Threshold value of 4.



Ridged MultiFractal example using Lattice Convolution Noise and a Threshold value of 2.

Here is the network. It contains one Ridged MultiFractal node, a Gradient node, and a Clamp node.

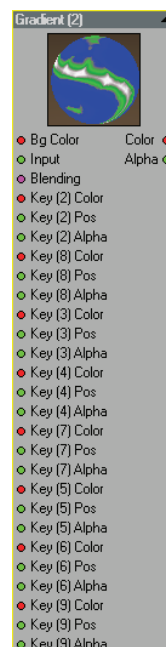


The Clamp node is being used to remove any details or texture noise that may be present in the lower texture values that are being used as the ocean surface.

Expanding on this network model can create much more accurate and eye pleasing terrains. In order to achieve such an expansion double click on the Gradient node and check the Show Output option box for each one of the Keys that define the gradient.

With the Gradient node in its expanded state access to the Key Colors and the Key Alpha values become available for texturing.

Here is what the gradient node looks like expanded.



This will enable you to connect different textures for each elevation level (gradient key). This can be achieved by adding texture nodes to the network and connecting their Color outputs to the Key [n] Color inputs. Additionally you should be able to create more realistic blends between the various textures that you have added by using the Key [n] Alpha inputs on the Gradient node.

Realism can be taken one step further by applying an appropriately subtle texture to the gradient nodes Key [n] Pos inputs. This step may require an Add node and/or a Multiply node per Key [n] Pos connection in order to offset each of the key positions appropriately maintaining the overall Start and End positions (0 to 1) of the Gradient definition.

It's important to note that in order to use the Displacement input on the Surface destination node Enable Bump in the Deform tab of the Object Properties panel must be checked and a Distance value greater than zero must be entered.

For this example the Enable Bump box was checked and 15cm was entered as the Distance value. A value of 20 was entered as the Renderer SubPatch Per Object Level.



## Ripples

### Description

A procedural texture that produces a three dimensional ripple pattern.

### Inputs

#### Bg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the background color layer.

#### Fg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the foreground color layer.

#### Blending (type: integer):

Specifies one of the blending types (0~13) found in the blending type list at the beginning of this chapter.

Can receive input from other nodes in the network or can be specified to by selecting a blending mode from the Blending pull-down menu in the Edit Panel for the node.

For general use specifying the blending mode by using the Blending pull-down menu in the Edit Panel will probably be the most desirable method of use. However, for larger networks where control of the blending modes of many texture nodes simultaneously is needed or when conditional blending is desired then using a node connection to control this value may be advantageous.

#### Opacity (type: scalar):

Opacity is the amount that the selected blending mode is applied. You can think of opacity as the strength of the blend effect between the BG Color and the FG Color.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Wave Sources (type: scalar):

Defines the number of wave sources that interact with each other in order to create the overall Ripple texture effect.

Each wave source is a spherical ripple pattern. When a flat object surface for example, passes through the center of a wave source it appears very much alike to the effect one sees by throwing a stone into a body of very still water.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Wavelength (type: scalar):

Defines the wave length of the wave sources. This is the length or distance between the waves as measured peak to peak. This value is in ratio to the texture scale value meaning that if the texture scale was set to 1m for X, Y, and Z then a wavelength value of 1 would produce waves from each source exactly 1m apart. With the same scale if the wavelength value were set to 0.25 the results would be four waves per meter from each source as defined by the Wave Sources parameter.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Wave Speed (type: scalar):

Defines the speed of the wave in terms of distance from peak to peak per frame. For example given a Wave Length of 0.25 if the desired cycle point was at 60 frames into the animation the Wave Speed value would be 0.004166666667. This is the resulting number from dividing 0.25 by 60 (0.25/60).

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this value using the controls available in the Edit Panel for this node.

#### Function (type: function):

Connecting a function node to this input will transform the texture value of this texture based on the graph defined by the function thereby extending the range of this nodes producible patterns.

See the manual section on Function nodes for a description of how texture values can be transformed.

Can receive inputs from other function nodes in the network only.

#### Bump (type: vector):

Specifies a vector direction for modifying the surface normal. This is surface normal direction information and affects the way the surface is shaded. Care should be taken when connecting to this input. Connecting dissimilar types or non directional vectors may cause the surface to shade wrongly.

Can receive input from other nodes in the network only.

#### Bump Amplitude (type: scalar):

Specifies the bump height or "amplitude" of the Bump directional vectors.

Using Alpha or luminance (connecting Color) values from other textures in the network along with their respective bumps can have the effect of blending the bump textures together.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Since Bump Amplitude is a shading feature and not a displacement coordinate system value it is specified as a percentage. Specifically as a percentage of the texture value.



## Scale (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Scale receives its X, Y, Z, scale values either by entering them into the Scale tab in the Edit Panel or by connecting an input to Scale on the node in the Workspace.

Scale is the scale of the texture pattern that will be applied to your object surface.

If the World Coordinates checkbox is checked in the edit panel then Scale defines the scale of the texture in relation to the world coordinates of your scene. Also known as world coordinate 0,0,0, in X, Y, and Z respectively.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures scale is derived from the selected Reference Object keyframed Scale values.

Connections made in the workspace area will override any values entered by the user into the edit panel.

## Position (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Position receives its X, Y, Z, coordinate information either by entering it into the Position tab in the Edit Panel or by connecting an input to the Position connection on the node in the Workspace Area.

Position defines the center coordinate of the texture. By default these are offset values of the objects coordinates also sometimes referred to as the objects pivot point.

If the World Coordinates checkbox is checked in the Edit Panel then Position defines the center of the texture as offset values from the world coordinates of your scene. Also known as world coordinate 0,0,0, in X, Y, Z format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures position is derived from the selected Reference Objects keyframed Position values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Rotation (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Rotation receives H (Heading), P (Pitch), and B (Bank) rotational information either by entering degrees into the Rotation tab in the Edit Panel or as radians by connecting an input to the Rotation connection on the node in the Workspace Area.

Rotation defines the axial rotation around the textures center

point. By default these are offset values of the objects a rotational settings.

If the World Coordinates checkbox is checked in the Edit Panel then Rotation values define the texture rotation as offset values from the world coordinates of your scene which should be fixed. You can rely on your world coordinate rotation values to always be 0,0,0, in H, P, B format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures rotation is derived from the selected Reference Objects keyframed Rotation values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Outputs

### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

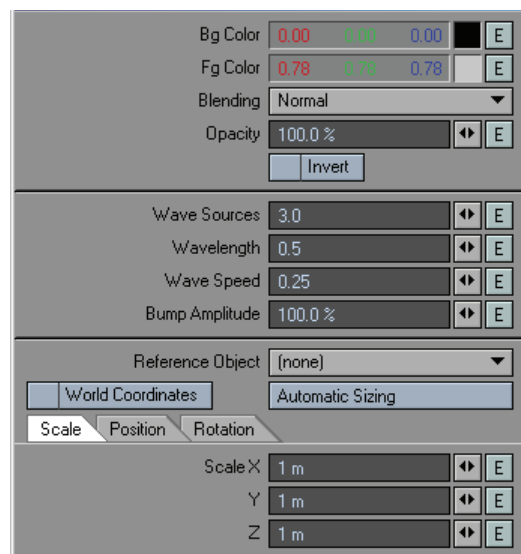
### Alpha (type: scalar):

Outputs single channel alpha (grey) information evaluated on a per spot basis.

### Bump (type: vector):

Outputs component vector direction information the length of which are the product of the bump amplitude value.

## Edit Panel



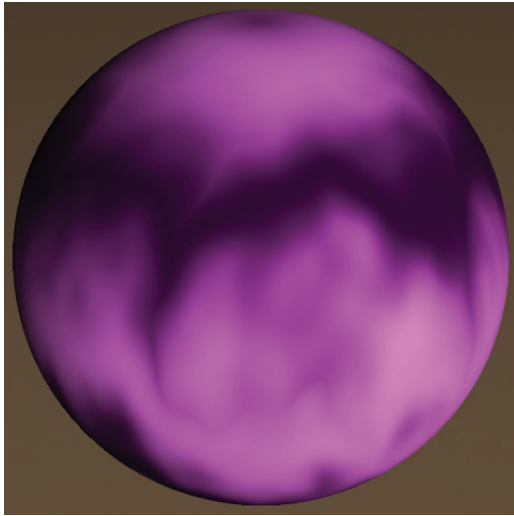
Reference Object, World Coordinates and Automatic Sizing are features exclusive to this nodes Edit Panel. Please see the explanation for these items in the chapter that covers texturing with the Surface Editor.





## Usage Example

Ripples liquid flow punctuated in pure purple - Animated.



A single frame of the rendered animation.

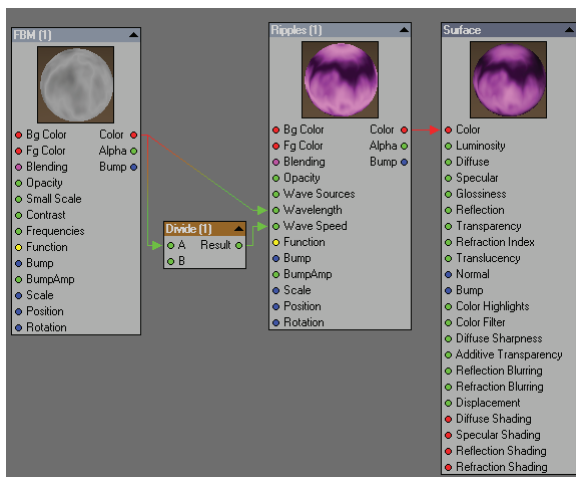
This network was set up to create a looping animation within 60 frames. It uses a single wave source which is deformed by connecting the output from the FBM Color to the Wave Length input of the Ripples node.

The divide node sets the Waves Speed which is actually a distance of value, to the same value being input as the Wavelength but divided by the number of frames where we want the cycle point to occur.

In this case our target animation is 60 frames so the input value is being divided by 60.

This was achieved simply by connecting the same input to both Wavelength and the Divide node, entering 60 as the B (divisor) value and connecting its Result to the Wave Speed input.

The Color output from the FBM node was used to achieve control over the value range simply by adjusting the color Values in the FBM Edit Panel.



## Turbulence

### Description

The general purpose fractalized noise texture, Turbulence is good for various aspects of naturally occurring phenomena such as fire, water, wind, smoke, clouds, or adding dirt, grime, and age to otherwise clean computer graphic looking surfaces.

### Inputs

#### Bg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the background color layer.

#### Fg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the foreground color layer.

#### Blending (type: integer):

Specifies one of the blending types (0~13) found in the blending type list at the beginning of this chapter.

Can receive input from other nodes in the network or can be specified to by selecting a blending mode from the Blending pull-down menu in the Edit Panel for the node.

For general use specifying the blending mode by using the Blending pull-down menu in the Edit Panel will probably be the most desirable method of use.

For larger networks where control of the blending modes of many texture nodes simultaneously is needed or when conditional blending is desired then using a node connection to control this value may be advantageous.

#### Opacity (type: scalar):

Opacity is the amount that the selected blending mode is applied. You can think of opacity as the strength of the blend effect between the BG Color and the FG Color.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.



## Small Scale (type: scalar):

Like Lacunarity, Small Scale is a counterpart to the fractal dimension that describes the texture of a fractal. It has to do with the size distribution of the gaps between detail levels. The Small Scale value sets the amount of change in scale between successive levels of detail.

If you're seeing repeating patterns in the texture it might be a good idea to alter the Small Scale value for the texture.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify values for this attribute using the controls available in the Edit Panel for this node.

## Contrast (type: scalar):

Sets the contrast of the texture value gradient. Higher values produce less gradient information between texture value extremes and therefore less noise is produced. The lower values spread the gradient more evenly between the extremes of the texture values and thus allow for more overall noise in the final output from this node.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify the values for this attribute using the controls available in the Edit Panel for this node.

## Frequencies (type: scalar):

"Frequencies" as used in this node is really just another name for octaves as defined in some of the descriptions of fractal type textures within this document.

You can think of "Frequencies" as the number of levels of detail mentioned in the discussion just above for Small Scale. At one "Frequency", only the basic pattern is used. At 2 "Frequencies" given a Small Scale value of 2, a one half scale pattern is overlaid on top of the basic pattern. And at three Frequencies, the overlays are one quarter scale over the one half scale over the basic and so on.

If Small Scale were 3 in this example, it would mean that each new iteration would be one third the scale of the previous layer.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify the values for this attribute using the controls available in the Edit Panel for this node.

## Function (type: function):

Connecting a function node to this input will transform the texture value of this texture based on the graph defined by the function thereby extending the range of this nodes producible patterns.

See the manual section on Function nodes for a description of how texture values can be transformed.

Can receive inputs from other function nodes in the network only.

## Bump (type: vector):

Specifies a vector direction for modifying the surface normal. This is surface normal direction information and affects the way the surface is shaded. Care should be taken when connecting to this input. Connecting dissimilar types or non directional vectors may cause the surface to shade wrongly.

Can receive input from other nodes in the network only.

## Bump Amplitude (type: scalar):

Specifies the bump height or "amplitude" of the Bump directional vectors.

Using Alpha or luminance (connecting Color) values from other textures in the network along with their respective bumps can have the effect of blending the bump textures together.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Since Bump Amplitude is a shading feature and not a displacement coordinate system value it is specified as a percentage. Specifically as a percentage of the texture value.

## Scale (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Scale receives its X, Y, Z, scale values either by entering them into the Scale tab in the Edit Panel or by connecting an input to Scale on the node in the Workspace.

Scale is the scale of the texture pattern that will be applied to your object surface.

If the World Coordinates checkbox is checked in the edit panel then Scale defines the scale of the texture in relation to the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, and Z respectively.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures scale is derived from the selected Reference Object keyframed Scale values.

Connections made in the workspace area will override any values entered by the user into the edit panel.

## Position (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Position receives its X, Y, Z, coordinate information either by entering it into the Position tab in the Edit Panel or by connecting an input to the Position connection on the node in the Workspace Area.

Position defines the center coordinate of the texture. By default these are offset values of the objects coordinates also sometimes referred to as the objects pivot point.





If the World Coordinates checkbox is checked in the Edit Panel then Position defines the center of the texture as offset values from the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, Z format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures position is derived from the selected Reference Objects keyframed Position values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Rotation (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Rotation receives H (Heading), P (Pitch), and B (Bank) rotational information either by entering degrees into the Rotation tab in the Edit Panel or as radians by connecting an input to the Rotation connection on the node in the Workspace Area.

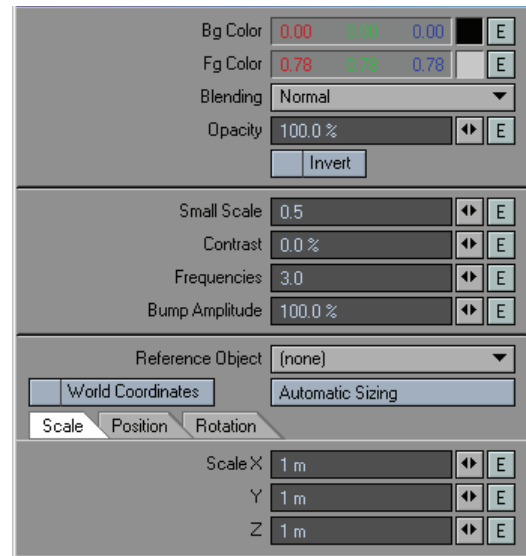
Rotation defines the axial rotation around the textures center point. By default these are offset values of the objects a rotational settings.

If the World Coordinates checkbox is checked in the Edit Panel then Rotation values define the texture rotation as offset values from the world coordinates of your scene which should be fixed. You can rely on your world coordinate rotation values to always be 0,0,0, in H, P, B format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures rotation is derived from the selected Reference Objects keyframed Rotation values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Edit Panel



Reference Object, World Coordinates and Automatic Sizing are features exclusive to this nodes Edit Panel.

## Outputs

### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

### Alpha (type: scalar):

Outputs single channel alpha (grey) information evaluated on a per spot basis.

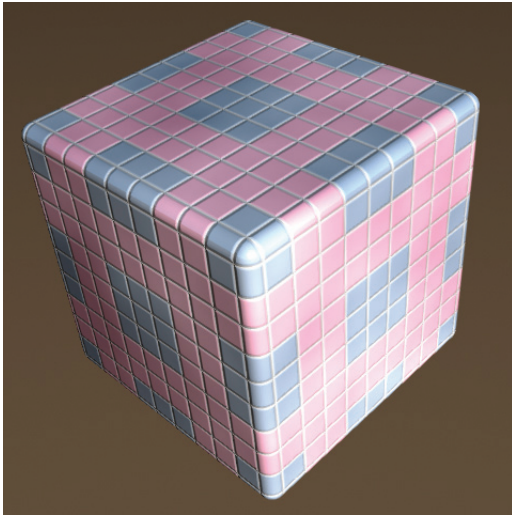
### Bump (type: vector):

Outputs component vector direction information the length of which are the product of the bump amplitude value.



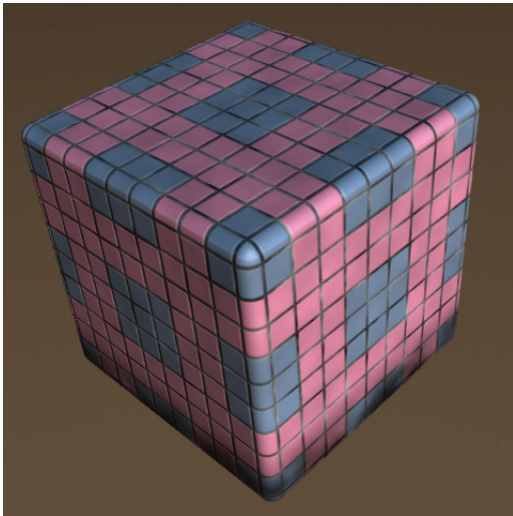
## Usage Example

Bleached out down and dirty with Turbulence.



Here is an example of the Turbulence texture being used to affect an object surface texture in order to cause it to appear Sun-bleached or perhaps the victim of overly strong detergents.

Be sure and check out the "3D Turbulence\_Example 2" preset from the "Manual\_Examples" Surface Preset [F8] library or load the "3DTurbulence\_Cube\_2.lwo" object, that were both installed with your LightWave3D content.



Here is the same object and surface as above but in this case Turbulence is being used to create a dirty grimy unwashed look.

Two Turbulence nodes are being used. Turbulence (1) is defining the dirt layer for the grout area set up in the Grid (1) node. As you can see from the preview sphere the Scale value has been set quite low - 10cm uniform on X, Y, and Z.

With a smaller Small Scale value of 4 and a slightly larger texture Scale value, Turbulence (2) is actually being used to define three separate surface attributes.

First its Color output is connected to the Surface destination nodes Diffuse input in order to create the overall dirty unwashed effect.

Secondly it's inverted Alpha is scaled by one third and used to define both the Specular and Glossiness attributes of the surface.

The inversion and scaling is achieved by connecting the Alpha output to an Invert node which is then connected to a divide node where 3 is entered for the B value (the divisor).

Finally a Gradient node is connected to the Turbulence (2) background color in order to make the bottom portion of the cube object appear even more grimy than the rest of the object surface.



Be sure and check out the "3D Turbulence\_Example 1" preset from the "Manual\_Examples" Surface Preset [F8] library or load the "3DTurbulence\_Cube\_1.lwo" object, that were both installed with your LightWave3D content.



## Turbulent Noise

### Description

A multipurpose turbulent noise texture good for a wide variety of applications.

### Inputs

#### Bg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the background color layer.

#### Fg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the foreground color layer.

#### Blending (type: integer):

Specifies one of the blending types (0~13) found in the blending type list at the beginning of this chapter.

Can receive input from other nodes in the network or can be specified to by selecting a blending mode from the Blending pull-down menu in the Edit Panel for the node.

For general use specifying the blending mode by using the Blending pull-down menu in the Edit Panel will probably be the most desirable method of use. However, for larger networks where control of the blending modes of many texture nodes simultaneously is needed or when conditional blending is desired then using a node connection to control this value may be advantageous.

#### Opacity (type: scalar):

Opacity is the amount that the selected blending mode is applied. You can think of opacity as the strength of the blend effect between the BG Color and the FG Color.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Increment (type: scalar):

Since this texture is a fractal or a fractalized noise pattern, it means that we can overlay smaller and smaller copies of the same pattern functions over previous layers in order to add detail. Technically, this is called Spectral Synthesis. These are not surface layers, but layers considered within the algorithm of the texture node itself.

Increment is the amplitude or intensity between successive levels of detail.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Lacunarity (type: scalar):

Lacunarity is a counterpart to the fractal dimension that describes the texture of a fractal. It has to do with the size distribution of the holes. Roughly speaking, if a fractal has large gaps or holes, it has high lacunarity; on the other hand, if a fractal is almost translationally invariant, it has low lacunarity. The lacunarity value sets the amount of change in scale between successive levels of detail.

If you're seeing repeating patterns in the texture it might be a good idea to lower the lacunarity level of the texture.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Octaves (type: scalar):

You can think of octaves as the number of levels of detail mentioned in the discussion just above for lacunarity. At one octave, only the basic pattern is used. At 2 octaves given a Lacunarity value of 2, a one half scale noise pattern is overlaid on top of the basic pattern. And at three octaves, the overlays are one quarter scale over the one half scale over the basic and so on.

If Lacunarity were 3 in this example, it would mean that each new layer would one third the scale of the previous layer.

Considering these three Increment, Lacunarity, and Octaves then if Lacunarity were set to 2, Increment were set to 0.5, and Octaves were 4, it would mean that each successive iteration for 4 iterations, was twice the frequency and half the amplitude.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.



## Function (type: function):

Connecting a function node to this input will transform the texture value of this texture based on the graph defined by the function thereby extending the range of this nodes producible patterns.

See the manual section on Function nodes for a description of how texture values can be transformed.

Can receive inputs from other function nodes in the network only.

## Bump (type: vector):

Specifies a vector direction for modifying the surface normal. This is surface normal direction information and affects the way the surface is shaded. Care should be taken when connecting to this input. Connecting dissimilar types or non directional vectors may cause the surface to shade wrongly.

Can receive input from other nodes in the network only.

## Bump Amplitude (type: scalar):

Specifies the bump height or "amplitude" of the Bump directional vectors.

Using Alpha or luminance (connecting Color) values from other textures in the network along with their respective bumps can have the effect of blending the bump textures together.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Since Bump Amplitude is a shading feature and not a displacement coordinate system value it is specified as a percentage. Specifically as a percentage of the texture value.

## Scale (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Scale receives its X, Y, Z, scale values either by entering them into the Scale tab in the Edit Panel or by connecting an input to Scale on the node in the Workspace.

Scale is the scale of the texture pattern that will be applied to your object surface.

If the World Coordinates checkbox is checked in the edit panel then Scale defines the scale of the texture in relation to the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, and Z respectively.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures scale is derived from the selected Reference Object keyframed Scale values.

Connections made in the workspace area will override any values entered by the user into the edit panel.

## Position (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Position receives its X, Y, Z. coordinate information either by entering it into the Position tab in the Edit Panel or by connecting an input to the Position connection on the node in the Workspace Area.

Position defines the center coordinate of the texture. By default these are offset values of the objects coordinates also sometimes referred to as the objects pivot point.

If the World Coordinates checkbox is checked in the Edit Panel then Position defines the center of the texture as offset values from the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, Z format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures position is derived from the selected Reference Objects keyframed Position values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Rotation (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Rotation receives H (Heading), P (Pitch), and B (Bank) rotational information either by entering degrees into the Rotation tab in the Edit Panel or as radians by connecting an input to the Rotation connection on the node in the Workspace Area.

Rotation defines the axial rotation around the textures center point. By default these are offset values of the objects a rotational settings.

If the World Coordinates checkbox is checked in the Edit Panel then Rotation values define the texture rotation as offset values from the world coordinates of your scene which should be fixed. You can rely on your world coordinate rotation values to always be 0,0,0, in H, P, B format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures rotation is derived from the selected Reference Objects keyframed Rotation values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.



## Outputs

### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

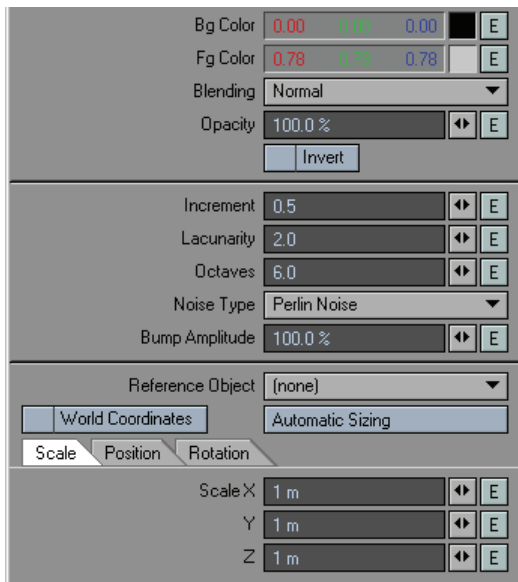
### Alpha (type: scalar):

Outputs single channel alpha (grey) information evaluated on a per spot basis.

### Bump (type: vector):

Outputs component vector direction information the length of which are the product of the bump amplitude value.

## Edit Panel



Reference Object, World Coordinates and Automatic Sizing are features exclusive to this nodes Edit Panel.

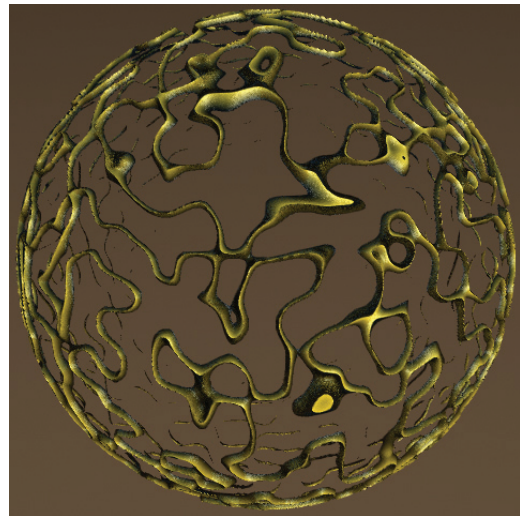
Please see the explanation for these items in the chapter that covers texturing with the Surface Editor.

The Noise Type selection popup menu also remains a feature exclusive to this nodes Edit Panel. Without going beyond the descriptions offered in the Surface Editor section of this manual for these related features, you can get a very good idea of what each Noise Type looks like by following these few simple steps:

1. Set the texture Scale values small enough that you can see the full pattern of the texture on the geometry you're working with.
2. Set Increment to zero.
3. Set Lacunarity to zero.
4. Set Octaves to zero.
5. Set the Offset value anywhere between 0.25 and 0.75
6. And select the various Noise Types one at a time while watching the VIPER preview update.

## Usage Example

Ancient gold scrolling with Turbulent Noise.



In this example turbulent noise is being used for a very wide range of surface attributes in order to create tarnished brass or gold scrolling.

There are two Turbulent Noise nodes in this example network. Turbulent Noise (2) is only being used to define very fine pits in the surface also commonly known as "micro-bump". Micro-bump enhances the specular and reflective properties of the surface by scattering reflected light which helps to achieve a more photo realistic surface.

The basic color for the gold or brass surface is defined by the foreground color of the Turbulent Noise (1) node.

The Alpha channel of the Turbulent Noise (1) is being used in two very distinct ways.

Firstly, it's connected to the Ceil node which for a default Alpha output will produce only zero or one values. The Ceil node rounds up all non integer values so that any numbers between zero and one become one and any numbers that are equal to zero remain zero.

The floating point range for an unmodified Alpha channel is usually zero through one (0~1).

The results of ceiling an Alpha from a Turbulent Noise node with a low Lacunarity value and is just what we want in order to apply non-gradient values it to the specular, glossiness, reflection and transparency inputs only where we want them.



NOTE: To see the results of this operation load the preset for this example and disconnect all connections from the Surface destination node. Now connect just the Ceil Out to the Diffuse input on the Surface destination node. To restore the example just reload the preset once again.

Since after the Ceil operation our values for the scrolling area are 1, we need to reduce them in order to get more realistic and acceptable values for our destination nodes input.



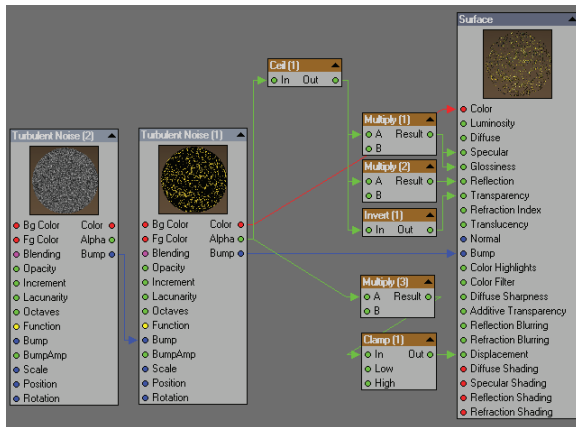


The first Multiply node multiplies the incoming values by 0.3 giving us 30% for Specular and Glossiness.

The second Multiply node multiplies the incoming values by 0.2 giving us 20% for the Reflection.

The all black or all white (0 or 1) Alpha is also inverted and used for the Transparency attribute of the surface.

Lastly, the Alpha output is multiplied and clamped in order to produce displacement values that are gradient for the scroll structure areas and flat for the areas which are transparent.



Remember to note that in order to use the Displacement input on the Surface destination node Enable Bump in the Deform tab of the Object Properties panel must be checked and a Distance value greater than zero must be entered.

For this example the Enable Bump box was checked and 15cm was entered as the Distance value. A value of 35 was entered as the Renderer SubPatch Per Object Level.

Be sure and check out the "3D Turbulent Noise\_Example" preset from the "Manual\_Examples" Surface Preset [F8] library or load the "3DTurbulentNoise\_Ball.lwo" object, that were both installed with your LightWave3D content.

## Underwater

### Description

A procedural texture that produces double interlocking rebel patterns per wave source in order to approximate wavy water refraction patterns.

### Inputs

#### Bg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the background color layer.

#### Fg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the foreground color layer.

#### Blending (type: integer):

Specifies one of the blending types (0~13) found in the blending type list at the beginning of this chapter.

Can receive input from other nodes in the network or can be specified to by selecting a blending mode from the Blending pull-down menu in the Edit Panel for the node.

For general use specifying the blending mode by using the Blending pull-down menu in the Edit Panel will probably be the most desirable method of use. However, for larger networks where control of the blending modes of many texture nodes simultaneously is needed or when conditional blending is desired then using a node connection to control this value may be advantageous.

#### Opacity (type: scalar):

Opacity is the amount that the selected blending mode is applied. You can think of opacity as the strength of the blend effect between the BG Color and the FG Color.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Wave Sources (type: scalar):

Defines the number of wave sources that interact with each other in order to create the overall Ripple texture effect.

Each wave source is a spherical double ripple pattern. When a flat object surface for example, passes through the center of a wave source it appears very much alike to the effect one sees by throwing a stone into a body of very still water.





Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

### Wavelength (type: scalar):

Defines the wave length of the wave sources. This is the length or distance between the double wave pattern. This value is in ratio to the texture scale value meaning that if the texture scale was set to 1m for X, Y, and Z then a wavelength value of 1 would produce waves from each source exactly 0.5m apart. The reason we arrive at the value of 0.5 is because as mentioned this is a double wave pattern - meaning two concentric waves.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

### Wave Speed (type: scalar):

Defines the speed of the waves in terms of distance from peak to peak per frame. For example given a wave length of 0.25 if the desired cycle point was at 60 frames into the animation the Wave Speed value would be 0.004166666667. This is the resulting number from dividing 0.25 by 60 (0.25/60).

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this value using the controls available in the Edit Panel for this node.

### Band Sharpness (type: scalar):

Controls the sharpness of the gradient between wave peaks and valleys.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this value using the controls available in the Edit Panel for this node.

### Function (type: function):

Connecting a function node to this input will transform the texture value of this texture based on the graph defined by the function thereby extending the range of this nodes producible patterns.

See the manual section on Function nodes for a description of how texture values can be transformed.

Can receive inputs from other function nodes in the network only.

### Bump (type: vector):

Specifies a vector direction for modifying the surface normal. This is surface normal direction information and affects the way the surface is shaded.

Care should be taken when connecting to this input. Connecting dissimilar types or non directional vectors may cause the surface to shade wrongly.

Can receive input from other nodes in the network only.

### Bump Amplitude (type: scalar):

Specifies the bump height or "amplitude" of the Bump directional vectors.

Using Alpha or luminance (connecting Color) values from other textures in the network along with their respective bumps can have the effect of blending the bump textures together.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Since Bump Amplitude is a shading feature and not a displacement coordinate system value it is specified as a percentage. Specifically as a percentage of the texture value.

### Scale (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Scale receives its X, Y, Z, scale values either by entering them into the Scale tab in the Edit Panel or by connecting an input to Scale on the node in the Workspace.

Scale is the scale of the texture pattern that will be applied to your object surface.

If the World Coordinates checkbox is checked in the edit panel then Scale defines the scale of the texture in relation to the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, and Z respectively.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures scale is derived from the selected Reference Object keyframed Scale values.

Connections made in the workspace area will override any values entered by the user into the edit panel.

### Position (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Position receives its X, Y, Z, coordinate information either by entering it into the Position tab in the Edit Panel or by connecting an input to the Position connection on the node in the Workspace Area.

Position defines the center coordinate of the texture. By default these are offset values of the objects coordinates also sometimes referred to as the objects pivot point.

If the World Coordinates checkbox is checked in the Edit Panel then Position defines the center of the texture as offset values from the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, Z format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures position is derived from the selected Reference Objects keyframed Position values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.



## Rotation (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Rotation receives H (Heading), P (Pitch), and B (Bank) rotational information either by entering degrees into the Rotation tab in the Edit Panel or as radians by connecting an input to the Rotation connection on the node in the Workspace Area.

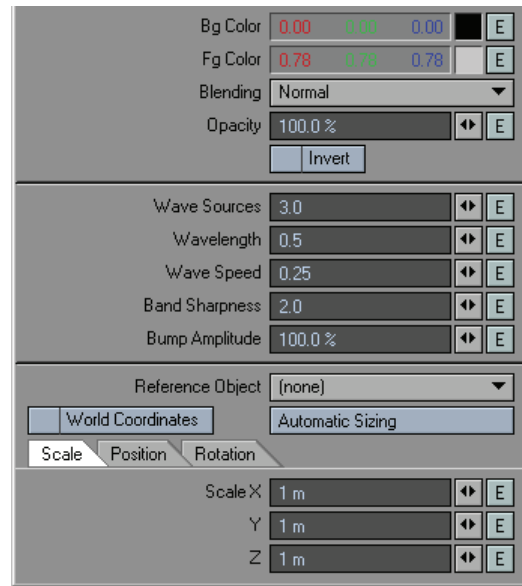
Rotation defines the axial rotation around the textures center point. By default these are offset values of the objects a rotational settings.

If the World Coordinates checkbox is checked in the Edit Panel then Rotation values define the texture rotation as offset values from the world coordinates of your scene which should be fixed. You can rely on your world coordinate rotation values to always be 0,0,0, in H, P, B format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures rotation is derived from the selected Reference Objects keyframed Rotation values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Edit Panel



Reference Object, World Coordinates and Automatic Sizing are features exclusive to this nodes Edit Panel. Please see the explanation for these items in the chapter that covers texturing with the Surface Editor.

## Outputs

### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

### Alpha (type: scalar):

Outputs single channel alpha (grey) information evaluated on a per spot basis.

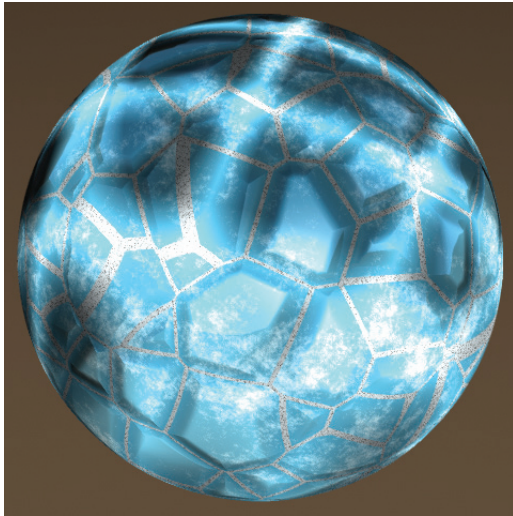
### Bump (type: vector):

Outputs component vector direction information the length of which are the product of the bump amplitude value.



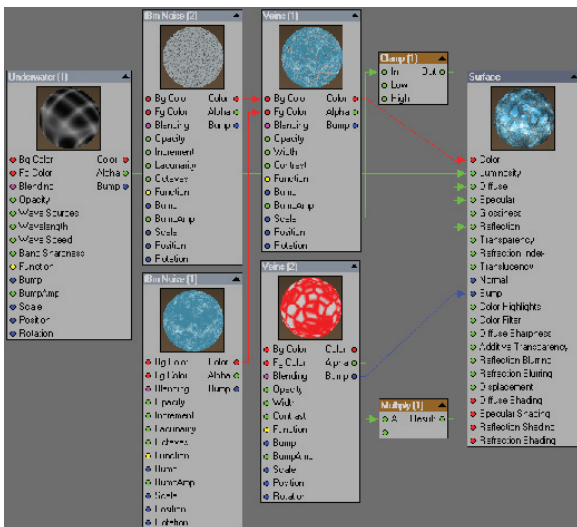
## Usage Example

Underwater refraction caustics faked on pond stones using the underwater node - Animated.



In this example the Underwater node is used on the luminosity channel in order to create a fairly convincing underwater light refraction pattern on an already existing texture.

As you can see this node network takes an existing rock pattern created with Veins explained in the Veins Usage Example, and simply applies the Underwater nodes Alpha output to the Luminosity of the surface destination node.



This technique can be used with almost any of the textures you may already have in order to add the appearance of that surface being underwater.

Be sure to check out the "3D Underwater\_Example" preset from the "Manual\_Examples" Surface Preset [F8] library or load the "3DUnderwater\_Ball.lwo" object, that were both installed with your LightWave3D content.

## Veins

### Description

A three dimensional cellular type texture that produces vein-like texture patterns.

### Inputs

#### Bg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the background color layer.

#### Fg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the foreground color layer.

#### Blending (type: integer):

Specifies one of the blending types (0~13) found in the blending type list at the beginning of this chapter.

Can receive input from other nodes in the network or can be specified to by selecting a blending mode from the Blending pull-down menu in the Edit Panel for the node.

For general use specifying the blending mode by using the Blending pull-down menu in the Edit Panel will probably be the most desirable method of use. However, for larger networks where control of the blending modes of many texture nodes simultaneously is needed or when conditional blending is desired then using a node connection to control this value may be advantageous.

#### Opacity (type: scalar):

Opacity is the amount that the selected blending mode is applied. You can think of opacity as the strength of the blend effect between the BG Color and the FG Color.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Width (type: scalar):

Controls the width in percentage, of the veins. Smaller values produce narrower sharper veins.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Contrast (type: scalar):

Defines the amount of contrast in percentage values, between the vein areas and the spaces in between the veins.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.



## Function (type: function):

Connecting a function node to this input will transform the texture value of this texture based on the graph defined by the function thereby extending the range of this nodes producible patterns.

See the manual section on Function nodes for a description of how texture values can be transformed.

Can receive inputs from other function nodes in the network only.

## Bump (type: vector):

Specifies a vector direction for modifying the surface normal. This is surface normal direction information and affects the way the surface is shaded. Care should be taken when connecting to this input. Connecting dissimilar types or non directional vectors may cause the surface to shade wrongly.

Can receive input from other nodes in the network only.

## Bump Amplitude (type: scalar):

Specifies the bump height or "amplitude" of the Bump directional vectors.

Using Alpha or luminance (connecting Color) values from other textures in the network along with their respective bumps can have the effect of blending the bump textures together.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Since Bump Amplitude is a shading feature and not a displacement coordinate system value it is specified as a percentage. Specifically as a percentage of the texture value.

## Scale (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Scale receives its X, Y, Z, scale values either by entering them into the Scale tab in the Edit Panel or by connecting an input to Scale on the node in the Workspace.

Scale is the scale of the texture pattern that will be applied to the object surface.

If the World Coordinates checkbox is checked in the edit panel then Scale defines the scale of the texture in relation to the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, and Z respectively.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures scale is derived from the selected Reference Object keyframed Scale values.

Connections made in the workspace area will override any values entered by the user into the edit panel.

## Position (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Position receives its X, Y, Z, coordinate information either

by entering it into the Position tab in the Edit Panel or by connecting an input to the Position connection on the node in the Workspace Area.

Position defines the center coordinate of the texture. By default these are offset values of the objects coordinates also sometimes referred to as the objects pivot point.

If the World Coordinates checkbox is checked in the Edit Panel then Position defines the center of the texture as offset values from the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, Z format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures position is derived from the selected Reference Objects keyframed Position values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Rotation (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Rotation receives H (Heading), P (Pitch), and B (Bank) rotational information either by entering degrees into the Rotation tab in the Edit Panel or as radians by connecting an input to the Rotation connection on the node in the Workspace Area.

Rotation defines the axial rotation around the textures center point. By default these are offset values of the objects a rotational settings.

If the World Coordinates checkbox is checked in the Edit Panel then Rotation values define the texture rotation as offset values from the world coordinates of your scene which should be fixed. You can rely on your world coordinate rotation values to always be 0,0,0, in H, P, B format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures rotation is derived from the selected Reference Objects keyframed Rotation values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Outputs

### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

### Alpha (type: scalar):

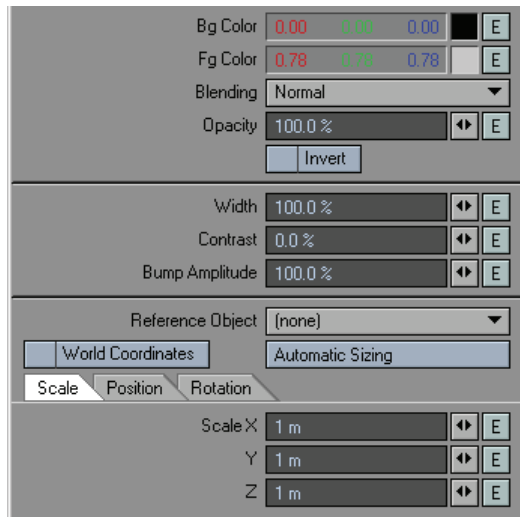
Outputs single channel alpha (grey) information evaluated on a per spot basis.

### Bump (type: vector):

Outputs component vector direction information the length of which are the product of the bump amplitude value.



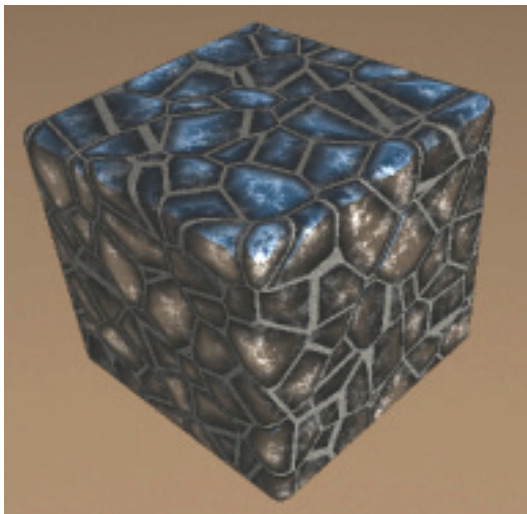
## Edit Panel



Reference Object, World Coordinates and Automatic Sizing are features exclusive to this node's Edit Panel. Please see the explanation for these items in the chapter that covers texturing with the Surface Editor.

## Usage Example

Garden pathway rocks using the Veins node.



This example render shows an inlaid stone texture using the Veins node, suitable for such surfaces as garden walkways, pool and pond bottoms, or ritzy hotel lobby walls.

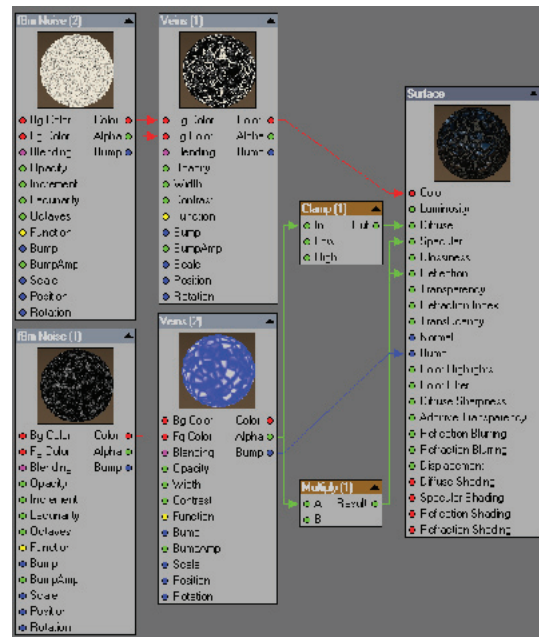
Achieving this result requires some manipulation of the default Vein node settings and its outputs.

Two vein nodes are used in this network. One for the reflective surface properties and bump map. The other to define the surface color.

An fBm Noise node was used for the foreground and background colors of the Veins node that defines the surface colors.

A very fine evenly detailed fBm Noise setting was used to color the actual vein areas and in combination with a clamped (flat) diffuse level, produces what looks very much like mortar areas between the inlaid rock.

A sparser rougher version of fBm Noise was used for the space in between the veins and together with the bump map and the Alpha output supplied from the Veins (2) node looks very much like a highly polished inlaid rock surface.



The Clamp node is used in this network in order to flatten the bottom trough areas of the otherwise gradient Veins (2) Alpha output. Had this not been added the grout areas between the rocks would appear almost completely black and produce an entirely different surface look.

The Multiply node is being used to adjust the range from the Alpha output of the Veins (2) node to bring the value range from between 0 (black) and 1 (white) to between 0 (black) and 0.3 (30% grey). This is a convenient method of reducing the intensity or "value range" from an Alpha channel.





## Wood

### Description

A three dimensional procedural texture that produces wood like ring patterns. This texture is not intended to produce a “photo-real” wood texture by itself alone but rather supply the user with the basic framework of a wood texture allowing users the freedom to add the kind of grain patterns and other details needed to produce the desired results. Wood offers the ability to scale the transition gradient ramp of the rings.

### Inputs

#### Bg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the background color layer.

#### Fg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the foreground color layer.

#### Blending (type: integer):

Specifies one of the blending types (0~13) found in the blending type list at the beginning of this chapter.

Can receive input from other nodes in the network or can be specified to by selecting a blending mode from the Blending pull-down menu in the Edit Panel for the node.

For general use specifying the blending mode by using the Blending pull-down menu in the Edit Panel will probably be the most desirable method of use.

For larger networks where control of the blending modes of many texture nodes simultaneously is needed or when conditional blending is desired then using a node connection to control this value may be advantageous.

#### Opacity (type: scalar):

Opacity is the amount that the selected blending mode is applied. You can think of opacity as the strength of the blend effect between the BG Color and the FG Color.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Frequencies (type: scalar):

Defines the number of noise frequencies (0~31) used to affect the basic ring pattern. Each frequency introduces compound affecting noise patterns.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Turbulence (type: scalar):

Defines the amount of influence each frequency will have on the basic ring pattern. This setting is in ratio of to texture Scale value.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Ring Spacing (type: scalar):

Defines the spacing value between rings in terms of layout units usually meters. As opposed to some other seemingly similar textures the Wood texture Scale value has no affect on ring spacing.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Ring Sharpness (type: scalar):

Controls the position of the end of the gradient ramp from the foreground color or ring start, to the background color also offset from the ring start edge.

You can think of each concentric ring as having a highly defined leading edge or “start” position and the Ring Sharpness as defining the thickness of each ring where the thickness is always a gradient from the foreground to the background color.

Higher values produce narrower rings while lower values produce wider rings. A value of about 4 will produce a gradient ramp length that ends in about the middle of the ring before the affecting Turbulence is applied.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Function (type: function):

Connecting a function node to this input will transform the texture value of this texture based on the graph defined by the function thereby extending the range of this nodes producible patterns.

See the manual section on Function nodes for a description of how texture values can be transformed.

Can receive inputs from other function nodes in the network only.





## Bump (type: vector):

Specifies a vector direction for modifying the surface normal. This is surface normal direction information and affects the way the surface is shaded. Care should be taken when connecting to this input. Connecting dissimilar types or non directional vectors may cause the surface to shade wrongly.

Can receive input from other nodes in the network only.

## Bump Amplitude (type: scalar):

Specifies the bump height or “amplitude” of the Bump directional vectors.

Using Alpha or luminance (connecting Color) values from other textures in the network along with their respective bumps can have the effect of blending the bump textures together.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Since Bump Amplitude is a shading feature and not a displacement coordinate system value it is specified as a percentage. Specifically as a percentage of the texture value.

## Scale (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Scale receives its X, Y, Z, scale values either by entering them into the Scale tab in the Edit Panel or by connecting an input to Scale on the node in the Workspace.

Scale is the scale of the texture pattern that will be applied to your object surface.

If the World Coordinates checkbox is checked in the edit panel then Scale defines the scale of the texture in relation to the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, and Z respectively.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures scale is derived from the selected Reference Object keyframed Scale values.

Connections made in the workspace area will override any values entered by the user into the edit panel.

## Position (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Position receives its X, Y, Z, coordinate information either by entering it into the Position tab in the Edit Panel or by connecting an input to the Position connection on the node in the Workspace Area.

Position defines the center coordinate of the texture. By default these are offset values of the objects coordinates also sometimes referred to as the objects pivot point.

If the World Coordinates checkbox is checked in the Edit Panel then Position defines the center of the texture as offset values from the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, Z format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures position is derived from the selected Reference Objects keyframed Position values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Rotation (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Rotation receives H (Heading), P (Pitch), and B (Bank) rotational information either by entering degrees into the Rotation tab in the Edit Panel or as radians by connecting an input to the Rotation connection on the node in the Workspace Area.

Rotation defines the axial rotation around the textures center point. By default these are offset values of the objects a rotational settings.

If the World Coordinates checkbox is checked in the Edit Panel then Rotation values define the texture rotation as offset values from the world coordinates of your scene which should be fixed. You can rely on your world coordinate rotation values to always be 0,0,0, in H, P, B format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures rotation is derived from the selected Reference Objects keyframed Rotation values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Outputs

### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

### Alpha (type: scalar):

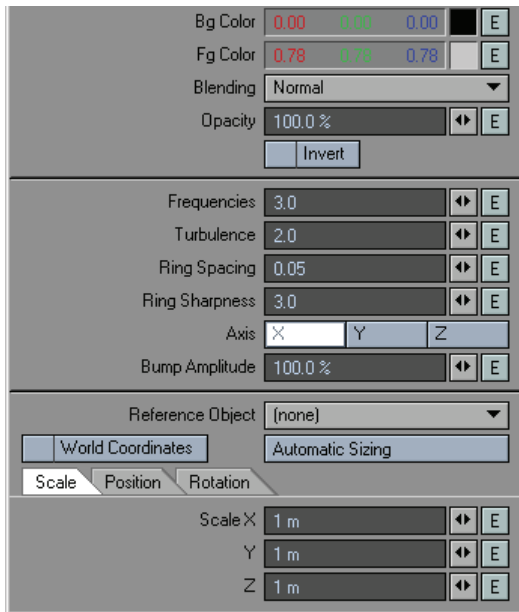
Outputs single channel alpha (grey) information evaluated on a per spot basis.

### Bump (type: vector):

Outputs component vector direction information the length of which are the product of the bump amplitude value.



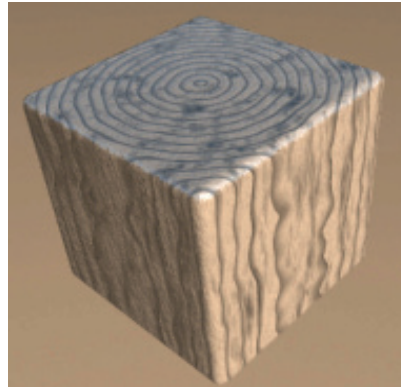
## Edit Panel



Axis are not offered as node connections in the Workspace area. Likewise Reference Object, World Coordinates and Automatic Sizing our features exclusive to this nodes Edit Panel. please see the explanation for these items in the chapter that covers texturing with the Surface Editor.

## Usage Example

More wood is better wood. Using gradient vectors for rotation and position.



In this example network three Wood nodes an fBm Noise node and a Turbulent Noise node are used to achieve a more detailed wood texture than is possible with a single Wood node.

As can be seen from the network example below two copies of the Wood (1) node are being used on the foreground color of each of the noise textures respectively.

The Wood (3) Bump output is connected to the fBm Noise Bump input which is mixing that result with the Wood (1) Bump. From this series of connections the bump channels from three individual nodes are mixed and finally connected to the surface Destination node.



What is strangely interesting about this example is that the Bump outputs from Wood nodes (2) and (3) are being used as the Position and Rotation vectors for the respective noise nodes. This had the pleasantly unexpected result of conforming the majority of the noise values produced by the noise nodes they are connected to, along the general wood pattern producing an effect that looked more like wood grain than without those connections in place.



## Wood2

### Description

A three dimensional texture that produces a wood like pattern of concentric rings. Wood2 offers the ability to phase the transition gradient ramp of the rings.

### Inputs

#### Bg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the background color layer.

#### Fg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the foreground color layer.

#### Blending (type: integer):

Specifies one of the blending types (0~13) found in the blending type list at the beginning of this chapter.

Can receive input from other nodes in the network or can be specified to by selecting a blending mode from the Blending pull-down menu in the Edit Panel for the node.

For general use specifying the blending mode by using the Blending pull-down menu in the Edit Panel will probably be the most desirable method of use. However, for larger networks where control of the blending modes of many texture nodes simultaneously is needed or when conditional blending is desired then using a node connection to control this value may be advantageous.

#### Opacity (type: scalar):

Opacity is the amount that the selected blending mode is applied. You can think of opacity as the strength of the blend effect between the BG Color and the FG Color.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Ring Spacing (type: scalar):

Defines the ring spacing in relationship to the texture Scale value. For example with a scale value of 1m for X, Y, and Z, a Ring Spacing value of 0.5 would result in a ring every 50cm or 2 rings per meter.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Phase (type: scalar):

The Phase value phases or "moves" the gradient transition in respect to the ring inner (start) and outer (end) edges.

Smaller values place the background to foreground gradient ramp start nearer the inner edge of the ring. Larger values place the gradient start nearer the outer edge of the ring.

A Phase value of 0.5 will center the gradient peak between the beginning and end of the ring width.

The length of the gradient ramp is equal to the span of a single ring.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Distortion (type: scalar):

Defines the amount of influence that Noise Scale may introduce to the shape of the rings.

Depending on the value of the noise scale this has the effect of "distorting" the otherwise concentric circular shape of the rings.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.



## Noise Scale (type: scalar):

Defines the scale of the single noise pattern.

Smaller values result in a smaller noise scale.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

## Function (type: function):

Connecting a function node to this input will transform the texture value of this texture based on the graph defined by the function thereby extending the range of this nodes producible patterns.

See the manual section on Function nodes for a description of how texture values can be transformed.

Can receive inputs from other function nodes in the network only.

## Bump (type: vector):

Specifies a vector direction for modifying the surface normal. This is surface normal direction information and affects the way the surface is shaded. Care should be taken when connecting to this input. Connecting dissimilar types or non directional vectors may cause the surface to shade wrongly.

Can receive input from other nodes in the network only.

## Bump Amplitude (type: scalar):

Specifies the bump height or "amplitude" of the Bump directional vectors.

Using Alpha or luminance (connecting Color) values from other textures in the network along with their respective bumps can have the effect of blending the bump textures together.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Since Bump Amplitude is a shading feature and not a displacement coordinate system value it is specified as a percentage. Specifically as a percentage of the texture value.

## Scale (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Scale receives its X, Y, Z, scale values either by entering them into the Scale tab in the Edit Panel or by connecting an input to Scale on the node in the Workspace.

Scale is the scale of the texture pattern that will be applied to your object surface.

If the World Coordinates checkbox is checked in the edit panel then Scale defines the scale of the texture in relation to the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, and Z respectively.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures scale is derived from the selected Reference Object keyframed Scale values.

Connections made in the workspace area will override any values entered by the user into the edit panel.

## Position (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Position receives its X, Y, Z. coordinate information either by entering it into the Position tab in the Edit Panel or by connecting an input to the Position connection on the node in the Workspace Area.

Position defines the center coordinate of the texture. By default these are offset values of the objects coordinates also sometimes referred to as the objects pivot point.

If the World Coordinates checkbox is checked in the Edit Panel then Position defines the center of the texture as offset values from the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, Z format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures position is derived from the selected Reference Objects keyframed Position values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Rotation (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Rotation receives H (Heading), P (Pitch), and B (Bank) rotational information either by entering degrees into the Rotation tab in the Edit Panel or as radians by connecting an input to the Rotation connection on the node in the Workspace Area.

Rotation defines the axial rotation around the textures center point. By default these are offset values of the objects a rotational settings.

If the World Coordinates checkbox is checked in the Edit Panel then Rotation values define the texture rotation as offset values from the world coordinates of your scene which should be fixed. You can rely on your world coordinate rotation values to always be 0,0,0. in H, P, B format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures rotation is derived from the selected Reference Objects keyframed Rotation values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.



## Outputs

### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

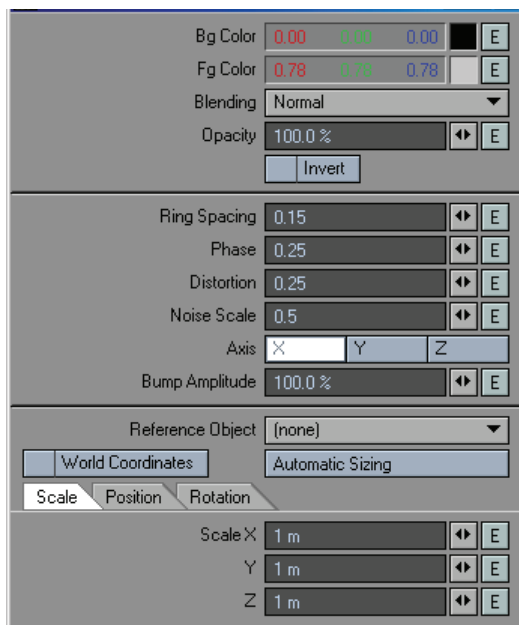
### Alpha (type: scalar):

Outputs single channel alpha (grey) information evaluated on a per spot basis.

### Bump (type: vector):

Outputs component vector direction information the length of which are the product of the bump amplitude value.

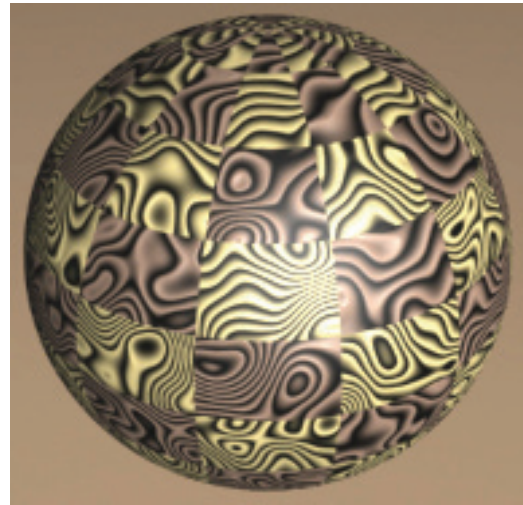
## Edit Panel



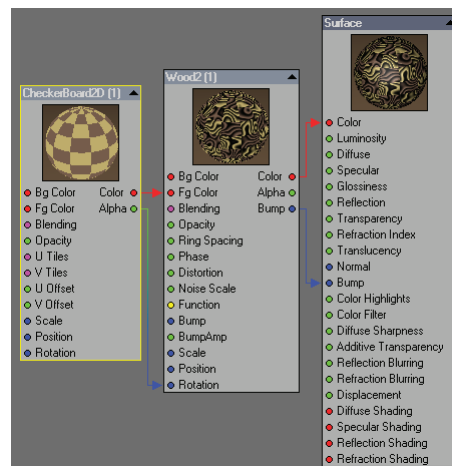
Axis are not offered as node connections in the Workspace area. Likewise Reference Object, World Coordinates and Automatic Sizing are features exclusive to this node's Edit Panel. Please see the explanation for these items in the chapter that covers texturing with the Surface Editor.

## Usage Example

A very simple example showing the default Wood2 node.



Here the Checkerboard2D node is being used to colorize the Wood2 texture and at the same time rotate the wood texture in such a way that each "block" looks like a slightly different wood texture.



Using the checkerboard to rotate the blocks is the most interesting aspect of this example.

Remember that radians are used for units of rotation for connected inputs in the Workspace area. Since our checkerboard Alpha map is solid white (1) and solid black (0) only, then only two rotational values are ever interpreted from the Alpha channel when we plug it in to the Rotation input of the Wood2 node.

Since 1 (one) radian is equal to about 57.17° and 0 (zero) radians are equal to 0° then wherever the incoming Alpha values are one the wood texture will be rotated approximately 57.17° on all three axis H, P, and B. And wherever the Alpha value is zero the wood texture will be rotated 0°.

Be sure to check out the "3D Wood2\_Example" preset from the "Manual\_Examples" Surface Preset [F8] library or load the "3DWood2\_Cube.lwo" object, that were both installed with your LightWave3D content.



## fBm Noise

### Description

A fractional Brownian motion (fBm) fractal noise texture. The results of several scientific studies have shown that many real world textures have the same kind of spectra noise as produced by fBm making it a natural choice for various kinds of naturally occurring texture noises and patterns.

### Inputs

#### Bg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the background color layer.

#### Fg Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color of the foreground color layer.

#### Blending (type: integer):

Specifies one of the blending types (0~13) found in the blending type list at the beginning of this chapter.

Can receive input from other nodes in the network or can be specified to by selecting a blending mode from the Blending pull-down menu in the Edit Panel for the node.

For general use specifying the blending mode by using the Blending pull-down menu in the Edit Panel will probably be the most desirable method of use.

However, for larger networks where control of the blending modes of many texture nodes simultaneously is needed or when conditional blending is desired then using a node connection to control this value may be advantageous.

#### Opacity (type: scalar):

Opacity is the amount that the selected blending mode is applied. You can think of opacity as the strength of the blend effect between the BG Color and the FG Color.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Increment (type: scalar):

Since this texture is a fractal or a fractalized noise pattern, it means that we can overlay smaller and smaller copies of the same pattern functions over previous layers in order to add detail. Technically, this is called Spectral Synthesis. These are not

surface layers, but layers considered within the algorithm of the texture node itself.

Increment is the amplitude or intensity between successive levels of detail.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Lacunarity (type: scalar):

Lacunarity is a counterpart to the fractal dimension that describes the texture of a fractal. It has to do with the size distribution of the holes. Roughly speaking, if a fractal has large gaps or holes, it has high lacunarity; on the other hand, if a fractal is almost translationally invariant, it has low lacunarity. The lacunarity value sets the amount of change in scale between successive levels of detail.

If you're seeing repeating patterns in the texture it might be a good idea to lower the lacunarity level of the texture.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Octaves (type: scalar):

You can think of octaves as the number of levels of detail mentioned in the discussion just above for lacunarity. At one octave, only the basic pattern is used. At 2 octaves given a Lacunarity value of 2, a one half scale noise pattern is overlaid on top of the basic pattern. And at three octaves, the overlays are one quarter scale over the one half scale over the basic and so on.

If Lacunarity were 3 in this example, it would mean that each new layer would one third the scale of the previous layer.

Considering these three Increment, Lacunarity, and Octaves then if Lacunarity were set to 2, Increment were set to 0.5, and Octaves were 4, it would mean that each successive iteration for 4 iterations, was twice the frequency and half the amplitude.

Can receive patterns and numerical inputs from other nodes in the network. The user may also specify this parameter value using the controls available in the Edit Panel for this node.

#### Function (type: function):

Connecting a function node to this input will transform the texture value of this texture based on the graph defined by the function thereby extending the range of this nodes producible patterns.

See the manual section on Function nodes for a description of how texture values can be transformed.

Can receive inputs from other function nodes in the network only.





## Bump (type: vector):

Specifies a vector direction for modifying the surface normal. This is surface normal direction information and affects the way the surface is shaded. Care should be taken when connecting to this input. Connecting dissimilar types or non directional vectors may cause the surface to shade wrongly.

Can receive input from other nodes in the network only.

## Bump Amplitude (type: scalar):

Specifies the bump height or “amplitude” of the Bump directional vectors.

Using Alpha or luminance (connecting Color) values from other textures in the network along with their respective bumps can have the effect of blending the bump textures together.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Since Bump Amplitude is a shading feature and not a displacement coordinate system value it is specified as a percentage. Specifically as a percentage of the texture value.

## Scale (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Scale receives its X, Y, Z, scale values either by entering them into the Scale tab in the Edit Panel or by connecting an input to Scale on the node in the Workspace.

Scale is the scale of the texture pattern that will be applied to your object surface.

If the World Coordinates checkbox is checked in the edit panel then Scale defines the scale of the texture in relation to the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, and Z respectively.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures scale is derived from the selected Reference Object keyframed Scale values.

Connections made in the workspace area will override any values entered by the user into the edit panel.

## Position (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Position receives its X, Y, Z, coordinate information either by entering it into the Position tab in the Edit Panel or by connecting an input to the Position connection on the node in the Workspace Area.

Position defines the center coordinate of the texture. By default these are offset values of the objects coordinates also sometimes referred to as the objects pivot point.

If the World Coordinates checkbox is checked in the Edit Panel then Position defines the center of the texture as offset values from the world coordinates of your scene. Also known as world coordinate 0,0,0. in X, Y, Z format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures position is derived from the selected Reference Objects keyframed Position values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Rotation (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Rotation receives H (Heading), P (Pitch), and B (Bank) rotational information either by entering degrees into the Rotation tab in the Edit Panel or as radians by connecting an input to the Rotation connection on the node in the Workspace Area.

Rotation defines the axial rotation around the textures center point. By default these are offset values of the objects a rotational settings.

If the World Coordinates checkbox is checked in the Edit Panel then Rotation values define the texture rotation as offset values from the world coordinates of your scene which should be fixed. You can rely on your world coordinate rotation values to always be 0,0,0, in H, P, B format.

If a Reference Object is assigned via the pull-down selection menu in the Edit Panel then the textures rotation is derived from the selected Reference Objects keyframed Rotation values.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Outputs

### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

### Alpha (type: scalar):

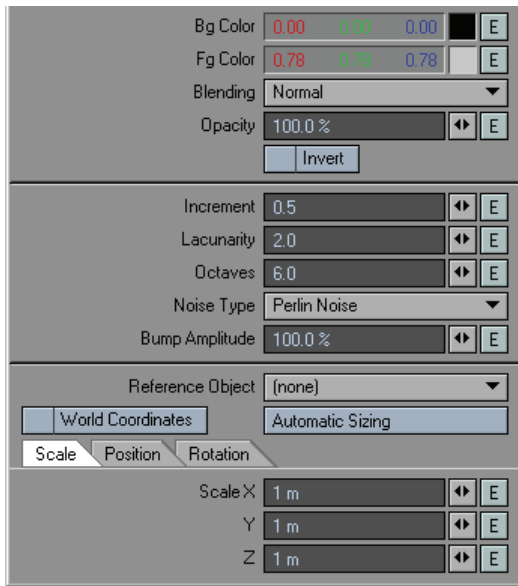
Outputs single channel alpha (grey) information evaluated on a per spot basis.

### Bump (type: vector):

Outputs component vector direction information the length of which are the product of the bump amplitude value.



## Edit Panel



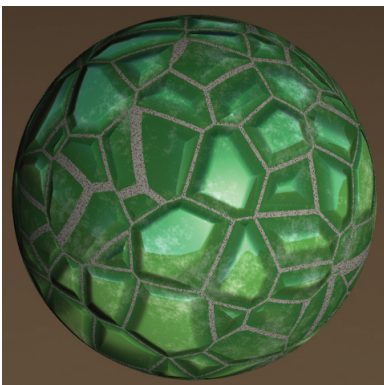
Reference Object, World Coordinates and Automatic Sizing are features exclusive to this nodes Edit Panel. Please see the explanation for these items in the chapter that covers texturing with the Surface Editor.

The Noise Type selection popup menu also remains a feature exclusive to this nodes Edit Panel. Without going beyond the descriptions offered in the Surface Editor section of this manual for these related features, you can get a very good idea of what each Noise Type looks like by following these few simple steps:

1. Set the texture Scale values small enough that you can see the full pattern of the texture on the geometry you're working with.
2. Set Increment to zero.
3. Set Lacunarity to zero.
4. Set Octaves to zero.
5. And select the various Noise Types one at a time while watching the VIPER preview update.

## Usage Example

Igneous fusion and pyrogenic strata formations (pretty rocks) using fBm Noise.



This example render shows and inlaid stone texture using the

Veins node, suitable for such surfaces as garden walkways, pond bottoms, or various stone inlay used in architectural construction.

An fBm Noise node was used for the foreground and background colors of the Veins node that defines the surface colors.

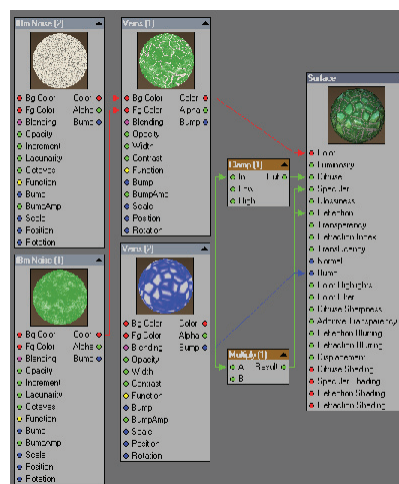
Two vein nodes are also used in this network. One for the reflective surface properties and bump gradient. The other to define the surface color.

A very fine evenly detailed fBm Noise setting was used to color the actual vein areas and in combination with a clamped (flat) diffuse level, produces what looks very much like mortar areas between inlaid rock.

A sparser rougher version of fBm Noise was used for the space in between the veins and together with the bump map and the Alpha output supplied from the Veins (2) node, looks very much like a highly polished inlaid rock surface.

The Clamp node is used in this network in order to flatten the bottom trough areas of the otherwise gradient ramp that is output from the Veins (2) Alpha output.

The Multiply node is being used to adjust the range from the Alpha output of the Veins (2) node to bring the value range of 0 (black) through 1 (white) to 0 (black) through 0.3 (30% grey). This is a very common method of reducing the intensity or "value range" from an Alpha channel.





## Constant

## Angle

### Purpose

Allows a user defined constant to be created to drive any angular input.

### Description

This is a standalone node capable of outputting a single angular value as defined by the user. This can be used to drive any number of angular inputs, allowing them to be set up and controlled from a single node.

### Inputs

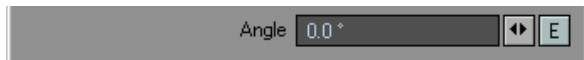
None

### Outputs

Angle (Type: Scalar):

Scalar output in degrees with a nominal range of 0 to 360 degrees.

### Edit Panel



### Usage Examples

Simply connect the output to any scalar input.

## Color

### Purpose

Allows a user defined constant to be created to drive any color input.

### Description

This is a standalone node capable of outputting a single color as defined by the user. This can be used to drive any number of color inputs, allowing them to be set up and controlled from a single node.

### Inputs

None

### Outputs

Color (Type: Color):

Standard RGB color output.

### Edit Panel



### Usage Examples

Simply connect the output to any color input.

## Direction

### Purpose

Allows a user defined constant to be created to drive any direction or rotation input.

### Description

This is a standalone node capable of outputting a direction vector as defined by the user. This can be used to drive any number of direction inputs, allowing them to be set up and controlled from a single node.

### Inputs

None

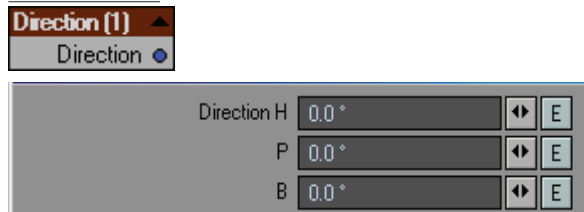


## Outputs

Vector (Type: Vector):

Vector output in degrees each with a nominal range of 0 to 360 degrees.

### Edit Panel



### Usage Examples

Attach the output from this node to the rotation input of any number of nodes to allow them to all be rotated in unison from a single point.

## Integer

### Purpose

Allows a user defined constant to be created to drive any integer input.

### Description

This is a standalone node capable of outputting an integer as defined by the user. This can be used to drive any number of integer inputs, allowing them to be set up and controlled from a single node.

### Inputs

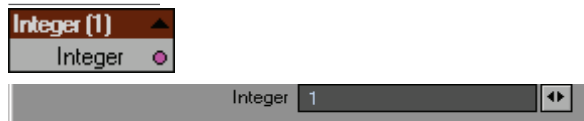
None

### Outputs

Integer (Type: Integer):

Integer output.

### Edit Panel



### Usage Examples

A good use for this node would be to add two integer constant nodes and connect them to the U Tile and V Tile inputs of all 2D nodes. This would allow the tiling of the entire texture to be controlled from two simple controls.

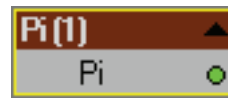
## Pi

### Purpose

Allows the constant Pi to drive another input.

### Description

The Pi node supplies a high precision of the constant Pi. Pi is derived from the fact that the ratio of a circle's circumference to its diameter is always constant.



### Inputs

None

### Outputs

Outputs a high precision value of Pi. This number: 3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706798214808651328230664709384460955058223172535940812848111745028410270193852110555964462294895493038196442881097566593344612847564823378678316527120190914564856692346034861045432664821339360726024914127372458700660631558817488152092096282925409171536436789259036001133053054882046652138414695194151160943305727036575959195309218611738193261179310511854807446237996274956735188575272489122793818301194912.

### Edit Panel

None

## Scalar

### Purpose

Allows a user defined constant to be created to drive any scalar input.

### Description

This is a standalone node capable of outputting a scalar as defined by the user. This can be used to drive any number of scalar inputs, allowing them to be set up and controlled from a single node.

### Inputs

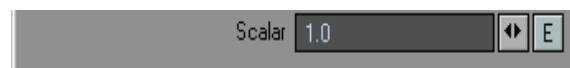
None

### Outputs

Scalar (Type: Scalar):

Scalar output.

### Edit Panel



### Usage Examples

Simply connect the output to any scalar input.



## Vector

### Purpose

Allows a user defined constant to be created to drive any vector input.

### Description

This is a standalone node capable of outputting a vector as defined by the user. This can be used to drive any number of vector inputs, allowing them to be set up and controlled from a single node.

### Inputs

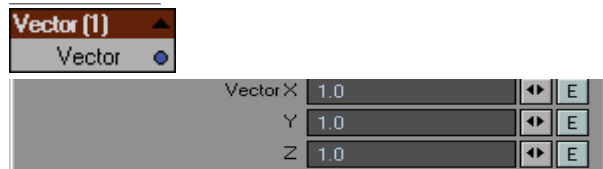
None

### Outputs

Vector (Type: Vector):

Vector output.

### Edit Panel



### Usage Examples

This node is useful for driving vector inputs such as Scale or Position and can control them from a single point.

## FUNCTIONS

### Bias

#### Purpose

The Bias function allows the brightness of the connected node to be controlled.

#### Description

This function node provides a standard bias function. Bias is a specific function that can be used to control the brightness on any texture with a function input. This function acts much like the brightness control on a television.



#### Inputs

Frequency (Type: Scalar):

The Frequency input controls the time taken to complete one function cycle. By increasing the frequency, the cycle period of the function is reduced. Likewise, decreasing the frequency means the function takes longer to complete one whole cycle. The mode (described below) determines the behavior of the function once it has completed a full cycle.

Amplitude (Type: Scalar):

The Amplitude input controls the size of the function from peak to peak.

Phase (Type: Scalar):

The Phase input allows the function waveform to be shifted in time. Increasing the phase shifts the function to the left, while decreasing it shifts the function to the right.



## Mode (Type: Constant, Repeat or Oscillate):

Three modes are available that determine how the function behaves after one complete cycle has been performed. The possible choices are:

- Constant mode, in which the function remains as its last value when one cycle completes.
- Repeat mode, in which the function restarts from the beginning when one cycle completes, giving a repeating oscillation based on a saw tooth waveform.
- Oscillate mode, in which the function completes one cycle and then returns to its start point over the next cycle. This provides a repeating oscillation based on a triangular waveform.

## Clamp Low (Type: Scalar):

The Clamp Low input defines a point at which, if the function is less than, the output is clipped. In other words, the output of the function can be no lower than this value.

## Clamp High (Type: Scalar):

The Clamp High input defines a point at which, if the function is greater than, the output is clipped. In other words, the output of the function can be no higher than this value.

## Bias (Type: Scalar):

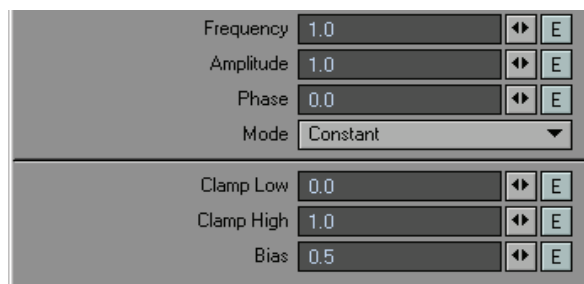
The Bias input controls the actual bias of the function. By increasing this value, the brightness of the attached node is increased while, conversely, reducing it reduces the brightness.

## Outputs

Result (Type: Function):

The function output must be connected to the function input of the texture you wish to control.

## Edit Panel



## Usage Examples

Connect the result output to the function input on any texture node (3D or 2D). By adjusting the Bias control you can then control the brightness of the attached node.

## BoxStep

### Purpose

The Box Step function allows a connected node to be clamped between two specified values.



### Description

This function node provides a box step function. A box step is a ramped step between two user specified values. This function effectively clips the connected texture, allowing flats and plateaus to be created.

### Inputs

#### Frequency (Type: Scalar):

The Frequency input controls the time taken to complete one function cycle. By increasing the frequency, the cycle period of the function is reduced. Likewise, decreasing the frequency means the function takes longer to complete one whole cycle. The mode (described below) determines the behavior of the function once it has completed a full cycle.

#### Amplitude (Type: Scalar):

The Amplitude input controls the size of the function from peak to peak.

#### Phase (Type: Scalar):

The Phase input allows the function waveform to be shifted in time. Increasing the phase shifts the function to the left, while decreasing it shifts the function to the right.





## Mode (Type: Constant, Repeat or Oscillate):

Three modes are available that determine how the function behaves after one complete cycle has been performed. The possible choices are:

- Constant mode, in which the function remains as its last value when one cycle completes.
- Repeat mode, in which the function restarts from the beginning when one cycle completes, giving a repeating oscillation based on a saw tooth waveform.
- Oscillate mode, in which the function completes one cycle and then returns to its start point over the next cycle. This provides a repeating oscillation based on a triangular waveform.

## Clamp Low (Type: Scalar):

The Clamp Low input defines a point at which, if the function is less than, the output is clipped. In other words the output of the function can be no lower than this value.

## Clamp High (Type: Scalar):

The Clamp High input defines a point at which, if the function is greater than, the output is clipped. In other words the output of the function can be no higher than this value.

## Begin (Type: Scalar):

The Begin input specifies the start of the step. That is, the position for the start point of the ramp up.

## End (Type: Scalar):

The End input specifies the finish of the step. That is, the position for the end point of the ramp up.

## Outputs

Result (Type: Function):

The function output must be connected to the function input of the texture you wish to control.

## Usage Examples

Connect the result output to the function input on any texture node (3D or 2D). By adjusting the Begin and End points, the valleys and peaks of the connected texture can be flattened.

## Gain



## Purpose

The Gain function allows the contrast of the connected node to be controlled.

## Description

This function node provides a standard gain function. Gain is a specific function that can be used to control the contrast on any texture with a function input. This function acts much like the contrast control on a television.

## Inputs

### Frequency (Type: Scalar):

The Frequency input controls the time taken to complete one function cycle. By increasing the frequency, the cycle period of the function is reduced. Likewise, decreasing the frequency means the function takes longer to complete one whole cycle. The mode (described below) determines the behavior of the function once it has completed a full cycle.

### Amplitude (Type: Scalar):

The Amplitude input controls the size of the function from peak to peak.

### Phase (Type: Scalar):

The Phase input allows the function waveform to be shifted in time. Increasing the phase shifts the function to the left, while decreasing it shifts the function to the right.

## Mode (Type: Constant, Repeat or Oscillate):

Three modes are available that determine how the function behaves after one complete cycle has been performed. The possible choices are:

- Constant mode, in which the function remains as its last value when one cycle completes.
- Repeat mode, in which the function restarts from the beginning when one cycle completes, giving a repeating oscillation based on a saw tooth waveform.
- Oscillate mode, in which the function completes one cycle then returns to its start point over the next cycle. This provides a repeating oscillation based on a triangular waveform.



## Clamp Low (Type: Scalar):

The Clamp Low input defines a point at which, if the function is less than, the output is clipped. In other words, the output of the function can be no lower than this value.

## Clamp High (Type: Scalar):

The Clamp High input defines a point at which, if the function is greater than, the output is clipped. In other words, in the output of the function can be no higher than this value.

## Gain (Type: Scalar):

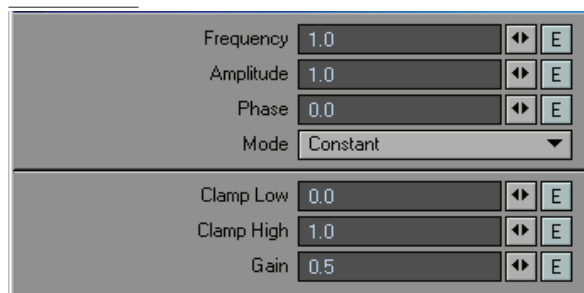
The Gain input controls the actual gain of the function. By increasing this value, the contrast of the attached node is increased while, conversely, reducing it reduces the contrast.

## Outputs

### Result (Type: Function):

The function output must be connected to the function input of the texture you wish to control.

## Edit Panel



## Usage Examples

Connect the result output to the function input on any texture node (3D or 2D). By adjusting the Gain control you can then control the contrast of the attached node.

## Gamma

### Purpose

The Gamma function allows the connected node to be gamma corrected.

### Description

This function node provides a standard gamma correction function. Gamma correction is a function that adjusts the brightness and contrast simultaneously.



### Inputs

#### Frequency (Type: Scalar):

The Frequency input controls the time taken to complete one function cycle. By increasing the frequency, the cycle period of the function is reduced. Likewise, decreasing the frequency means the function takes longer to complete one whole cycle. The mode (described below) determines the behavior of the function once it has completed a full cycle.

#### Amplitude (Type: Scalar):

The Amplitude input controls the size of the function from peak to peak.

#### Phase (Type: Scalar):

The Phase input allows the function waveform to be shifted in time. Increasing the phase shifts the function to the left, while decreasing it shifts the function to the right.

#### Mode (Type: Constant, Repeat or Oscillate):

Three modes are available that determine how the function behaves after one complete cycle has been performed. The possible choices are:

- Constant mode, in which the function remains as its last value when one cycle completes.
- Repeat mode, in which the function restarts from the beginning when one cycle completes, giving a repeating oscillation based on a saw tooth waveform.
- Oscillate mode, in which the function completes one cycle then returns to its start point over the next cycle. This provides a repeating oscillation based on a triangular waveform.



## Clamp Low (Type: Scalar):

The Clamp Low input defines a point at which, if the function is less than, the output is clipped. In other words the output of the function can be no lower than this value.

## Clamp High (Type: Scalar):

The Clamp High input defines a point at which, if the function is greater than, the output is clipped. In other words in the output of the function can be no higher than this value.

## Gamma (Type: Scalar):

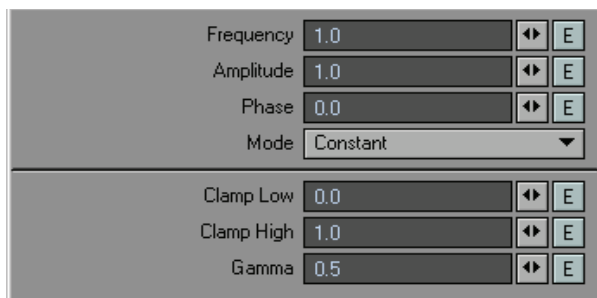
The Gamma input controls the overall gamma correction of the function. By increasing this value, the brightness and contrast of the attached node are increased while, conversely, reducing it reduces the brightness and contrast.

## Outputs

### Result (Type: Function):

The function output must be connected to the function input of the texture you wish to control.

## Edit Panel



## Usage Examples

Connect the result output to the function input on any texture node (3D or 2D). By adjusting the Gamma control you can then control the brightness and contrast of the attached node.

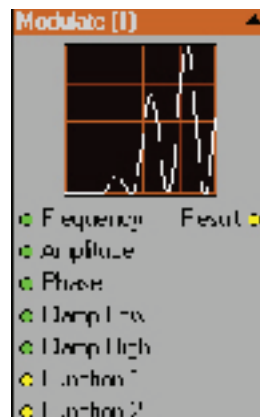
## Modulate

### Purpose

The Modulate node allows two functions to be combined in a variety of ways.

### Description

This function node provides a method that allows two other functions to be combined. The combination may be Add, Subtract, Multiply, Maximum, or Minimum. This node extends the tools available for creating new functions by combining two others.



### Inputs

#### Frequency (Type: Scalar):

The Frequency input controls the time taken to complete one function cycle. By increasing the frequency, the cycle period of the function is reduced. Likewise, decreasing the frequency means the function takes longer to complete one whole cycle. The mode (described below) determines the behavior of the function once it has completed a full cycle.

#### Amplitude (Type: Scalar):

The Amplitude input controls the size of the function from peak to peak.

#### Phase (Type: Scalar):

The Phase input allows the function waveform to be shifted in time. Increasing the phase shifts the function to the left, while decreasing it shifts the function to the right.

#### Mode (Type: Constant, Repeat or Oscillate):

Three modes are available that determine how the function behaves after one complete cycle has been performed. The possible choices are:

- Constant mode, in which the function remains as its last value when one cycle completes.
- Repeat mode, in which the function restarts from the beginning when one cycle completes giving a, repeating oscillation based on a saw tooth waveform.
- Oscillate mode, in which the function completes one cycle then returns to its start point over the next cycle. This provides a repeating oscillation based on a triangular waveform.



## Clamp Low (Type: Scalar):

The Clamp Low input defines a point at which, if the function is less than, the output is clipped. In other words the output of the function can be no lower than this value.

## Clamp High (Type: Scalar):

The Clamp High input defines a point at which, if the function is greater than, the output is clipped. In other words the output of the function can be no higher than this value.

## Function 1 & Function 2 (Type: Function):

These provide the inputs for the two functions to be combined.

## Mode (Type: Add, Multiply, Subtract, Maximum or Minimum):

The Mode input defines how the two input functions are combined to provide the output. The choices are:

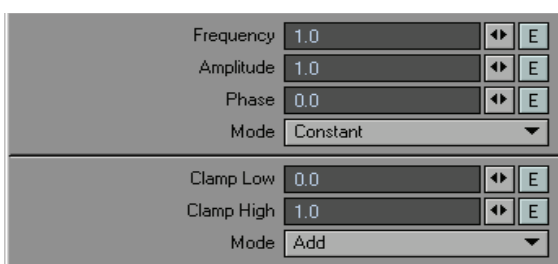
- Add, in which the output is the two input functions added.
- Subtract, in which the output is function input 1 minus function input 2.
- Multiply, in which the output is the two input functions multiplied.
- Maximum, in which the output is the maximum of the two input functions.
- Minimum, in which the output is the minimum of the two input functions.

## Outputs

### Result (Type: Function):

The function output must be connected to the function input of the texture you wish to control.

## Edit Panel



## Usage Examples

Connect the result output to the function input on any texture node (3D or 2D). Then connect the two function inputs (Function1 and Function2) to two different function nodes. The mode drop-down can then be used to combine these two functions in different ways.

## Noise

### Purpose

The Noise function allows the connected nodes texture blending to be disturbed.

### Description

This function node provides a standard noise function. The noise function is a standard Perlin noise function, which can be used disturb the blending or transition of the attached texture.



### Inputs

#### Frequency (Type: Scalar):

The Frequency input controls the time taken to complete one function cycle. By increasing the frequency, the cycle period of the function is reduced. Likewise, decreasing the frequency means the function takes longer to complete one whole cycle. The mode (described below) determines the behavior of the function once it has completed a full cycle.

#### Amplitude (Type: Scalar):

The Amplitude input controls the size of the function from peak to peak.

#### Phase (Type: Scalar):

The Phase input allows the function waveform to be shifted in time. Increasing the phase shifts the function to the left, while decreasing it shifts the function to the right.

#### Mode (Type: Constant, Repeat or Oscillate):

Three modes are available that determine how the function behaves after one complete cycle has been performed. The possible choices are:

- Constant mode, in which the function remains as its last value when one cycle completes.
- Repeat mode, in which the function restarts from the beginning when one cycle completes, giving a repeating oscillation based on a saw tooth waveform.
- Oscillate mode, in which the function completes one cycle then returns to its start point over the next cycle. This provides a repeating oscillation based on a triangular waveform.



## Clamp Low (Type: Scalar):

The Clamp Low input defines a point at which, if the function is less than, the output is clipped. In other words the output of the function can be no lower than this value.

## Clamp High (Type: Scalar):

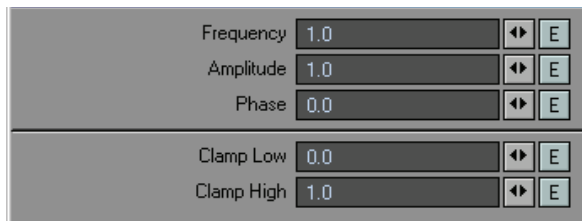
The Clamp High input defines a point at which, if the function is greater then, the output is clipped. In other words in the output of the function can be no higher than this value.

## Outputs

Result (Type: Function):

The function output must be connected to the function input of the texture you wish to control.

## Edit Panel



## Usage Examples

Connect the result output to the function input on any texture node (3D or 2D). You can now disturb and add noise to the transition of the attached texture.

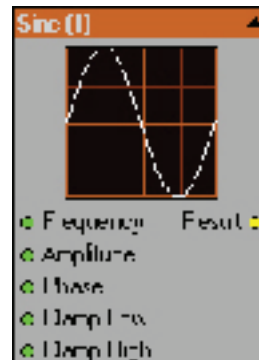
## Sine

### Purpose

The Sine function allows the connected nodes texture blending to be sine modulated.

### Description

This function node provides a standard sine function. This can then be used to modulate the blending of the attached texture. The sine function may also be shifted to provide a cosine as required.



### Inputs

#### Frequency (Type: Scalar):

The Frequency input controls the time taken to complete one function cycle. By increasing the frequency, the cycle period of the function is reduced. Likewise, decreasing the frequency means the function takes longer to complete one whole cycle. The mode (described below) determines the behavior of the function once it has completed a full cycle.

#### Amplitude (Type: Scalar):

The Amplitude input controls the size of the function from peak to peak.

#### Phase (Type: Scalar):

The Phase input allows the function waveform to be shifted in time. Increasing the phase shifts the function to the left, while decreasing it shifts the function to the right.

#### Mode (Type: Constant, Repeat or Oscillate):

Three modes are available that determine how the function behaves after one complete cycle has been performed. The possible choices are:

- Constant mode, in which the function remains as its last value when one cycle completes.
- Repeat mode, in which the function restarts from the beginning when one cycle completes, giving a repeating oscillation based on a saw tooth waveform.
- Oscillate mode, in which the function completes one cycle then returns to its start point over the next cycle. This provides a repeating oscillation based on a triangular waveform.



## Clamp Low (Type: Scalar):

The Clamp Low input defines a point at which, if the function is less than, the output is clipped. In other words the output of the function can be no lower than this value.

## Clamp High (Type: Scalar):

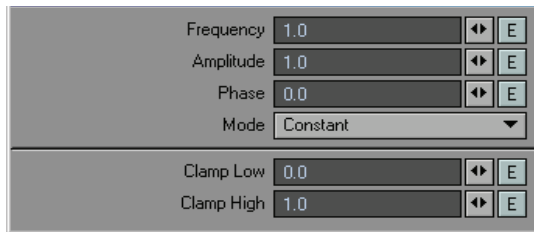
The Clamp High input defines a point at which, if the function is greater than, the output is clipped. In other words the output of the function can be no higher than this value.

## Outputs

### Result (Type: Function):

The function output must be connected to the function input of the texture you wish to control.

## Edit Panel



## Usage Examples

Connect the result output to the function input on any texture node (3D or 2D). You can now modulate the transition of the attached texture.

## SmoothStep

### Purpose

The Smooth Step function allows a connected node to be clamped and then smoothed between to user specified values.

### Description

This function node provides a smooth step function. A smooth step is a ramped step between two user specified values that has been smoothed so that the transition is more rounded. This function effectively clips the connected texture allowing flats and plateaus to be created, much like box step except the transitions are rounded. NOTE: This may be used in preference to the BoxStep function, as it provides automatic anti-aliasing.



### Inputs

#### Frequency (Type: Scalar):

The Frequency input controls the time taken to complete one function cycle. By increasing the frequency, the cycle period of the function is reduced. Likewise, decreasing the frequency means the function takes longer to complete one whole cycle. The mode (described below) determines the behavior of the function once it has completed a full cycle.

#### Amplitude (Type: Scalar):

The Amplitude input controls the size of the function from peak to peak.

#### Phase (Type: Scalar):

The Phase input allows the function waveform to be shifted in time. Increasing the phase shifts the function to the left, while decreasing it shifts the function to the right.

#### Mode (Type: Constant, Repeat or Oscillate):

Three modes are available that determine how the function behaves after one complete cycle has been performed. The possible choices are:

- Constant mode, in which the function remains as its last value when one cycle completes.
- Repeat mode, in which the function restarts from the beginning when one cycle completes, giving a repeating oscillation based on a saw tooth waveform.
- Oscillate mode, in which the function completes one cycle then returns to its start point over the next cycle. This provides a repeating oscillation based on a triangular waveform.





## Clamp Low (Type: Scalar):

The Clamp Low input defines a point at which, if the function is less than, the output is clipped. In other words the output of the function can be no lower than this value.

## Clamp High (Type: Scalar):

The Clamp High input defines a point at which, if the function is greater then, the output is clipped. In other words in the output of the function can be no higher than this value.

## Begin (Type: Scalar):

The Begin input specifies the start of the smooth step. That is, the position for the start point of the ramp up.

## End (Type: Scalar):

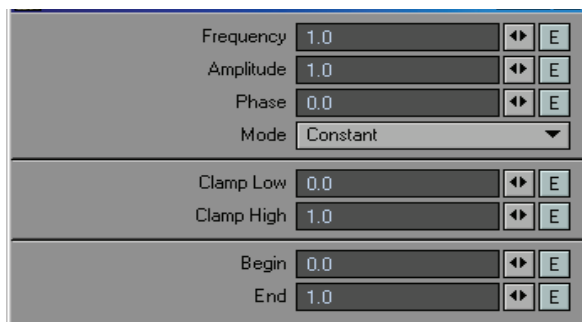
The End input specifies the finish of the smooth step. That is, the position for the end point of the ramp up.

## Outputs

### Result (Type: Function):

The function output must be connected to the function input of the texture you wish to control.

## Edit Panel



## Usage Examples

Connect the result output to the function input on any texture node (3D or 2D). By adjusting the Begin and End points, the valleys and peaks of the connected texture can be flattened and the transition smoothed.

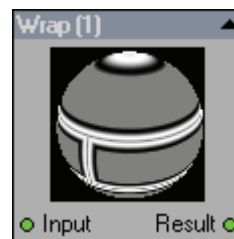
## Wrap

### Purpose

The Wrap node allows a scalar output to be post-processed using a function.

### Description

This node allows the scalar output from a texture to be modified by a function. Effectively, this allows the output from a texture to be transformed by a function without the need to use the dedicated function input. This makes it ideal for use when the final output is the result of a mathematical node.



### Inputs

#### Input Low (Type: Scalar):

A link only input. The scalar to be modified by the function should be connected to this input.

#### Function (Type: Function):

The function input. The function required to modify the scalar should be connected to this input.

### Outputs

#### Result (Type: Scalar):

Scalar output that is driven by the Scalar input as transformed by the connected function.

### Edit Panel

None.

## Usage Examples

Connect the scalar output from any texture into the Input node. Then connect any functions result into the Function input. The connected textures output will then be transformed using the connected function.



## Gradient

### Tools - Incidence

#### Purpose

The Incidence node derives the angle between the current hit point and the camera. It may also be used to derive the angle between the current hit point and any arbitrary vector.

#### Description

This node provides an output proportional to the angle between the current hit point and the camera. This makes it good for defining Fresnel kinds of effects. By driving in a user specified vector, the output is taken as the angle between that and the current hit point.

#### Inputs

**Vector** (Type: Vector):

By default, the Result output is defined as the angle between the viewing vector and the normal at the current hit point. However, any vector can be used to replace the viewing vector. For example, you may wish to drive the Vector input from a Make Vector node and define the input vector as 0.0, 1.0, and 0.0 for X, Y and Z respectively. This means that the Result output will be the angle between hit point and a vector defined as being straight up. The output will then appear as a "snow cap" for the current object.

**Normal** (Type: Vector):

The Normal input allows the angle to be perturbed. This input can be driven with any normal and is perturbed by the direction of that normal for the current hit point.

**Invert** (Type: Check Box):

The Invert check box, when enabled, inverts the result output. By default, the output goes from 1.0 facing to 0.0 glancing. When checked this is inverted.

**Range** (Type: 90 or 180):

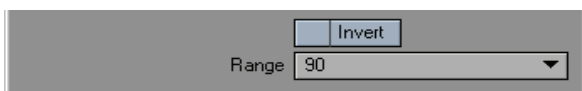
The Range selection allows the range of the angle to be changed from 90 degrees to 180 degrees. By default, the range is 90 degrees which means the value ranges between 0.0 for glancing polygons (faces at 90 degrees to the camera) and 1.0 for facing polygons (faces at 0 degrees to the camera). This may be changed to include faces that point away from the camera so their output will be 0.0 for faces that point away (faces at 180 degrees to the camera) and 1.0 for facing (faces at 0 degrees to the camera). All other angles are then interpolated between these.

#### Outputs

**Result** (Type: Scalar):

The Result output provides a normalized scalar output. That is, the result output varies between 0.0 and 1.0 proportional to the angle between the hit point and the camera (or Vector input).

#### Edit Panel



#### Usage Examples

Connect the Result output to a gradient and define a color gradient with a range 0.0 to 1.0. Then drive the gradient's Color output into the surface color.

## Tools - Thickness

#### Purpose

The Thickness node derives the thickness as defined by the length from the current hit point to the back of the object.

#### Description

This node provides a thickness function by firing a ray from the current hit point to the back of the object. This distance is output as the Length. For this node to function correctly, ray tracing must be enabled by checking any one of the four ray trace options (i.e. ray trace shadows, transparency, reflection or refraction) in the rendering options. The object must also have the double sided option enabled or all of the polygons correctly aligned so that the rays fired back to determine the thickness have something to hit.



#### Inputs

**IOR** (Type: Scalar):

The IOR input (Incidence Of Refraction) allows the rays fired to determine the thickness to be refracted. This happens in much the same way as light is refracted for transparency. If this input is 1.0, then the ray is fired straight back. As the value increases, the ray becomes more and more refracted.

**Normal** (Type: Vector):

The Normal input allows the rays fired back to be perturbed. This input can be driven with any normal and the rays fired back will be perturbed by the direction of that normal for the current hit point.

#### Outputs

**Length** (Type: Scalar):

This output provides the length from the hit point to the back of the object, i.e. the object's thickness for the current view.

**Hit Point** (Type: Vector):

This is the location where the ray hits.

**Direction** (Type: Scalar):

This is the direction of the ray.



## Edit Panel



## Usage Examples

Connect the Length output to a gradient and define a color gradient with a range based on the size of the object. Then drive the gradient's Color output into the surface color. Remember, ray tracing must be enabled and the object must be set-up so that the backside polygons point in the right direction.

## Gradient

### Purpose

The Gradient node allows a complex color and/or alpha gradient to be defined based on a series of key color/alpha values.

### Description

This node allows for either a color or an alpha gradient to be created. This gradient is then interpolated based on the input. The input may come from an external connected source such as a texture node or from one of the user selectable internal sources (e.g. Slope).

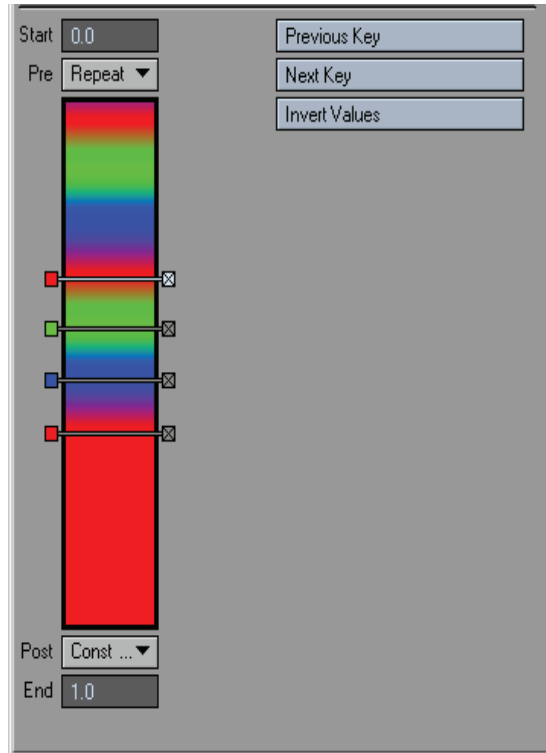
There are several aspects to the gradients and their creation. These are dealt with separately below:





## Basic Gradient Creation

In its simplest form, the gradient node allows a color gradient to be created. In its simplest form, the gradient node allows a color gradient to be created. In the bottom of the edit panel you can see a vertically aligned gradient. The range of the gradient is defined by the start and end values at the top and bottom of the gradient. These can be changed as required for different input ranges. For example, with the thickness you may wish to define a range from 0.0 to 10.0 meters. This can be achieved by editing these boxes.



You can add keys by clicking at the required insertion point on the gradient. Keys may be moved by clicking and dragging its associated square (on the left of each key). A single click on a key's square will select it. The currently selected key will be highlighted in a lighter color.

Keys may be deleted by clicking the corresponding X as seen on the right of the key.

You can also cycle through all the keys using the Previous Key and Next Key buttons.

The Invert button inverts (top to bottom) the values on all the keys in the gradient.

## Individual Key Editing



Once the range has been set and keys added, these can be edited individually using the control in the section above the gradient.

The Key text box will display the name of the currently selected key.

The Show Output check box allows the currently selected key to be added to the nodal inputs (i.e. the inputs will appear on the node and may then be connected to). When this is enabled Color, Position, and Alpha inputs are added for that key. This allows these parameters to be driven from other nodes so you could, for example, have the color for a key driven by the color output from a texture node.

The Color allows the color for that key to be defined. The output of the gradient is defined by the color of each key. Each key color can be enveloped and changed over time.

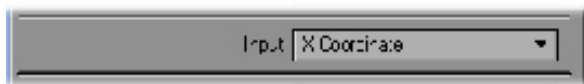
The Alpha allows the alpha value for that key to be defined. The alpha values primarily define how the current gradient is blended with the Bg Color. The blending is defined by the user controlled Blending input. The different blending modes are covered in the Reference Section on Blending Modes. Again, the alpha values may be enveloped and change over time.

The Position defines the position of the key on the gradient. For the key to appear on the gradient, the value must be between the start and end values (or equal to one of them). The key positions can be enveloped and changed over time. It is possible to provide a position outside the range of the gradient. This means that the key will not appear on the gradient and may only be selected using the Previous and Next Key selection buttons.

The Type defines the transition from the current key to the next. By default, this is a blend based on a Bezier curve. However, this also can be Linear for a straight blend, Step for a discrete step, or Hermite for a blend based on a Hermite curve.



## Driving the Gradient



The color output is defined by the gradient and some input. The input may be from an external node driving the Input node or from an internally selected gradient type. Essentially, the gradient provides a look-up table of colors. The input (internal or external) provides a position on the gradient. This gradient then uses the keys to interpolate an output color and alpha based on the input.

With nothing connected to external Input the gradient type may be selected using the Input drop down list. The choices are:

- The X Coordinate where the output is interpolated based on the gradient using the objects X coordinate for the current hit point as the input.
- The Y Coordinate where the output is interpolated based on the gradient using the objects Y coordinate for the current hit point as the input.
- The Z Coordinate where the output is interpolated based on the gradient using the objects Z coordinate for the current hit point as the input.
- The Slope where the output is interpolated based on the gradient using the angle between the current hit point and the camera's up direction as the input.
- The Incidence where the output is interpolated based on the gradient using the angle between the current hit point and the camera's viewing direction as the input.

When the external Input is connected to another node, these options become unavailable, and then the output is based on the gradient using the connected node as the input.

## The Background Color and Blending

As discussed in the above section on individual key editing, the alpha values allow the blending between the gradient and the Bg Color to be controlled. Using this method, layers of color can be built up by driving the output from one gradient into the Bg Color and using the alpha values to overlay and mix the current gradient.

By default, the blending is Normal and the alpha defines a linear mix between the background color and the current gradient output. This may be changed and different mixing or blending methods employed. These blending methods are standard across all nodes, and their description can be found in section Reference Section on Blending Modes.

## Inputs

### Bg Color (Type: Color):

The Bg Color input defines the background color. That is, the color used as the background when the gradient is being used to blend between its color and the background.

### Input (Type: Scalar):

The External Input node, when connected, is used as the input for the gradient. The output color and alpha are derived using the gradient based on this input. When connected, the internal gradient selection becomes unavailable.

### Blending (Type: Integer):

The Blend input allows the blending mode for the blend between the gradient and the Bg Color to be selected by an external node.

### Key(n) Color (Type: Color):

### Key(n) Pos (Type: Scalar):

### Key(n) Alpha (Type: Scalar):

If enabled, additional input will be added that allow the Color, Position, and Alpha for a key to be driven by external nodes.

## Outputs

### Color (Type: Color):

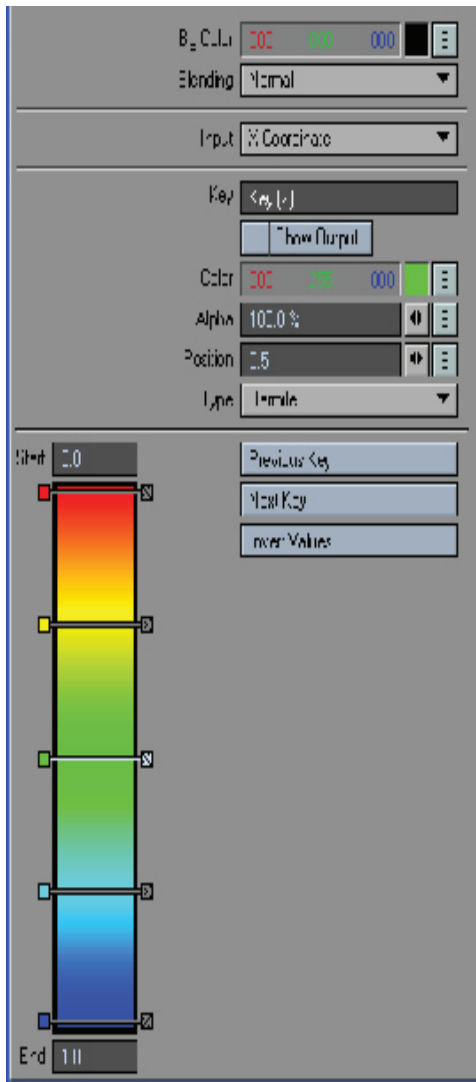
This is the Color output for the current gradient.

### Alpha (Type: Scalar):

This is the Alpha output as defined by the channel created by the alpha value for each key.



## Edit Panel



## Usage Examples

Create a color gradient with the default range 0.0 to 1.0 and connect the gradient Output to the surface color. Then, select one of the internal gradients as required. Alternatively, connect the Alpha output from texture into the gradient Input.

## ITEM INFO

### Camera

#### Purpose

The Camera node provides access to the current settings of the selected camera.

#### Description

This is a standalone node that provides access to a camera's current settings. Any of the cameras in the current scene may be selected. The outputs are then derived from the settings for the selected camera.



#### Inputs

Camera (Type: Camera Selection):

This drop down list allows any one of the available cameras in the scene to be selected. Once selected, the node's output will reflect the settings for that camera.

#### Outputs

Zoom Factor (Type: Scalar):

The Zoom Factor output is taken from the zoom factor as defined for the selected camera.

Focal Length (Type: Scalar):

The Focal Length output is taken from the lens focal length as defined for the selected camera.

Focal Distance (Type: Scalar):

Are these calculated?

Len F-Stop (Type: Scalar):

Are these calculated?





## Blur Length (Type: Scalar):

The Blur Length output is taken from the blur length as defined for the selected camera.

## Horizontal FOV (Type: Scalar):

The Horizontal FOV output is taken from the horizontal FOV as defined for the selected camera.

## Vertical FOV (Type: Scalar):

The Vertical FOV output is taken from the vertical FOV as defined for the selected camera.

## Width (Type: Integer):

The Width output is taken from the image width as defined for the selected camera.

## Height (Type: Integer):

The Height output is taken from the image height as defined for the selected camera.

## Pixel Width (Type: Scalar):

The Pixel Width output is taken from the pixel width ratio as defined for the selected camera.

## Eye Separation (Type: Scalar):

The Eye Separation output is taken from the eye separation as defined for the selected camera when stereoscopic rendering is enabled.

## Edit Panel



## Usage Examples

Simply connect any of the outputs as required. Bear in mind that the ranges for these outputs are not normalized and may vary with a range greater than 0.0 to 1.0.

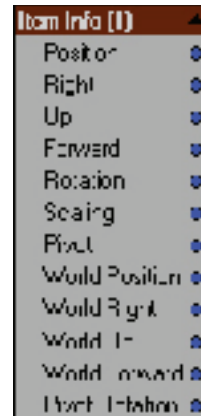
## Item Info

### Purpose

The Item Info node provides access to any of the item info settings defined for any item (Object, Camera, or Light) in the current scene.

### Description

This is a standalone node that provides access to any information contained within a certain item. The item may be an Object, Camera, or Light.



### Inputs

Item (Type: Item Selection):

This drop down list allows any one of the current scene's items to be selected. The selectable items are objects, cameras, or lights.

### Outputs

#### Position (Type: Vector):

The Position output defines the position of the selected item before parenting. If the selected item is parented, then this is the position relative to its parent.

#### Right (Type: Vector):

The Right output defines the direction vector of the X axis for the selected item in world coordinates.

#### Up (Type: Vector):

The Up output defines the direction vector of the Y axis for the selected item in world coordinates.

#### Forward (Type: Vector):

The Forward output defines the direction vector of the Z axis for the selected item in world coordinates.



## Rotation (Type: Vector):

The Rotation output defines the rotation of the selected item in radians. This is relative to the parent if the item is parented.

## Scaling (Type: Vector):

The Scaling output defines the scale factors of the selected item. This is relative to the parent if the item is parented.

## Pivot (Type: Vector):

The Pivot output defines the position of the item pivot point in the item's own coordinates. The pivot point is the origin of the item's rotation.

## World Position (Type: Vector):

The World Position output defines the position of the selected item in world coordinates after any parenting.

## World Right (Type: Vector):

The World Right output defines the direction vector of the X axis for the world in terms of the item's coordinates. This can be considered the inverse of the item's Right vector.

## World Up (Type: Vector):

The World Up output defines the direction vector of the Y axis for the world in terms of the item's coordinates. This can be considered the inverse of the item's Up vector.

## World Forward (Type: Vector):

The World Forward output defines the direction vector of the Z axis for the world in terms of the item's coordinates. This can be considered the inverse of the item's Forward vector.

## Pivot Rotation (Type: Vector):

The Pivot Rotation output defines the rotation the item's pivot point in radians. The pivot point is the origin of the item's rotation.

## Usage Examples

Add a NULL object and select it as the item in the Item Info node. Then connect item's Position output to the Center input of a texture node. The center of this texture will now track the NULL.

## Light Info

### Purpose

The Light Info node provides access to any settings defined for any light in the current scene.

### Description

This is a standalone node that provides access to any lights current settings. Any of the lights in the current scene may be selected. The outputs are then derived from the settings for the selected light.

### Inputs

Light (Type: Light Selection):

This drop down list allows any one of the current scene's lights to be selected.

### Outputs

Color (Type: Color):

The Color output is derived directly from the light color for the selected light.

Range (Type: Scalar):

The Range output is defined by the range setting for the selected light. If the selected light has no fall-off defined, then this is zero.

Intensity (Type: Scalar):

The Intensity output is defined by the intensity setting for the selected light.

Shadow Color (Type: Color):

The Shadow Color output is derived directly from the shadow color for the selected light.

Direction (Type: Vector):

The Direction output is taken as the direction in which the selected light is pointing. For non-directional lights, such as a point light, this is taken from the current alignment.

## Usage Examples

Connect any of the light info outputs to another node. For example, you could attach the Direction output to the Center input of a texture node. The texture would then move based on the direction of the selected light.



## Layers

### Bump Layer

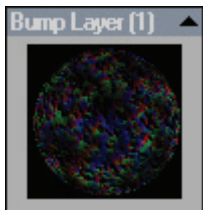
#### Purpose

The Bump Layer node allows access to the layered, bump, procedural textures, and gradients for integration into the nodal system.

#### Description

This node allows access to the standard layered procedural textures within LightWave. In this case, the procedural texture can be used for defining a bump gradient. The primary role of this component is to allow the integration of LightWave's layered procedural texture and gradient scheme into the nodal domain.

The interface for the Bump Layer node is the same as that for a standard LightWave texture layer. For more information on this, see reference LW Layer Texture Documentation. The only difference here is that the background for the texture is available as a connectible bump input and an alpha output is available as a scalar, based on the last texture layer's result.



#### Inputs

##### Bump (Type: Vector):

The Bump input is used as the base bump gradient for all the texture layers.

#### Outputs

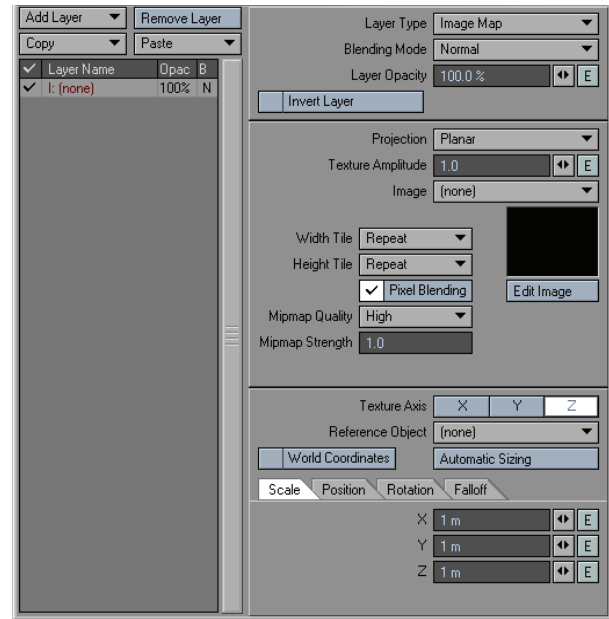
##### Bump (Type: Vector):

The Bump output is the result of all the defined procedural textures and gradient layers mixed with the input bump gradient.

##### Alpha (Type: Scalar):

The Alpha output is derived from the intensity of the last layer in the procedural texture/gradient stack.

## Edit Panel



## Usage Examples

Add the Bump Layer node and bring up its edit panel. Then, simply create a bump texture in the same way as you might using the standard LightWave layered approach. The output bump gradient can then drive into the final surface bump input.



## Color Layer

### Purpose

The Color Layer node allows access to the layered, color, procedural textures, and gradients for integration into the nodal system.

### Description

This node allows access to the standard layered procedural textures within LightWave. In this case, the procedural texture can be used for defining a color. The primary role of this component is to allow the integration of LightWave's layered procedural texture and gradient scheme into the nodal domain.

The interface for the Color Layer mode is the same as that for a standard LightWave texture layer. For more information on this, see reference LW Layer Texture Documentation. The only difference here is that the background for the texture is available as a connectible color input and an alpha output is available as a scalar, based on the last texture layer's result.



### Inputs

**Color** (Type: Color):

The Color input is used as the background color for all the texture layers.

### Outputs

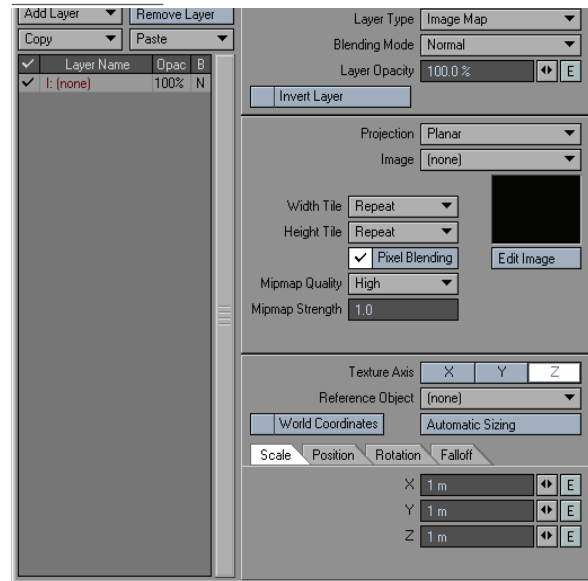
**Color** (Type: Color):

The Color output is the result of all the defined procedural texture and gradient layers mixed with the background color.

**Alpha** (Type: Scalar):

The Alpha output is derived from the intensity of the last layer in the procedural texture/gradient stack.

### Edit Panel



### Usage Examples

Add the Color Layer node and bring up its edit panel. Then, simply create a color texture in the same way as you might using the standard LightWave layered approach. The output color can then drive into the final surface color. You can also add background by adding another texture node and connecting its color output into the Color Layer's color input.



## Scalar Layer

### Purpose

The Scalar Layer node allows access to the layered procedural textures and gradients for integration into the nodal system.

### Description

This node allows access to the standard layered procedural textures within LightWave. In this case, the procedural texture can be used for defining a scalar. The primary role of this component is to allow the integration of LightWave's layered procedural texture and gradient scheme into the nodal domain.

The interface for the Scalar Layer node is the same as that for a standard LightWave texture layer. For more information on this, see reference LW Layer Texture Documentation. The only difference here is that the background for the texture is available as a connectible scalar input and an alpha output is available, based on the last texture layer's result.



### Inputs

Scalar (Type: Scalar):

The Scalar input is used as the base value or scalar for all the texture layers.

### Outputs

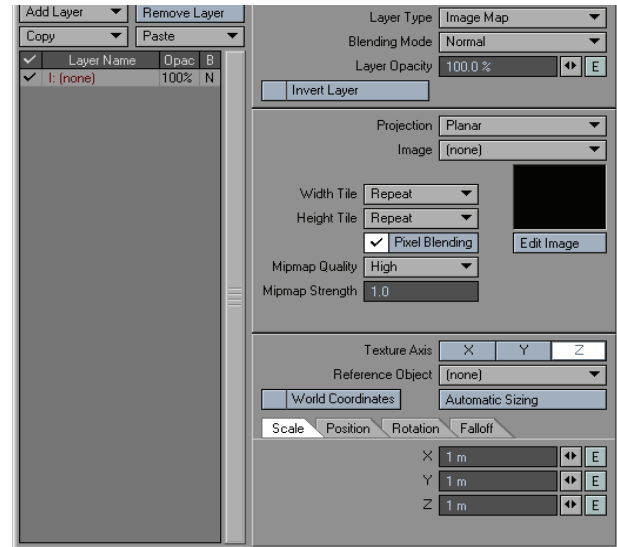
Scalar (Type: Scalar):

The Scalar output is the result of all the defined procedural textures and gradient layers mixed with the base scalar input.

Alpha (Type: Scalar):

The Alpha output is derived from the intensity of the last layer in the procedural texture/gradient stack.

## Edit Panel



## Usage Examples

Add the Scalar Layer node and bring up its edit panel. Then, simply create a scalar (alpha) texture in the same way as you might using the standard LightWave layered approach. The scalar output can then drive into the final surface scalar inputs like luminosity

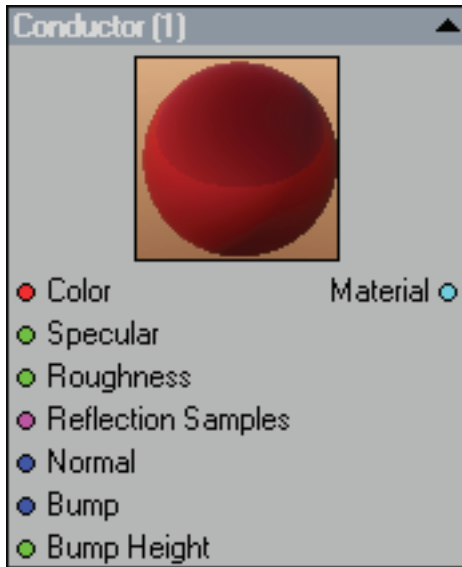


## Material Nodes

The Material nodes combine features of existing nodes into a more convenient and easier to use node system. The Material nodes duplicate specific surfaces, but can still access the power of the node system. Several types of material nodes are available.

### Conductor

Appropriate for simulating metallic surface finishes accurately. The appropriate values for various conductors (metals) can be obtained from published data in handbooks of optical constants.



### Inputs

#### BG Color

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

#### Specularity

Defines the amount of specularity that is applied to the surface by this node.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

#### Roughness

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Specifies the amount of roughness that will be applied. Roughness is the result of microfacet evaluation in lighting conditions. Higher values produce rougher surface properties while lower values produce smoother looking surfaces. This kind of roughness is more easily seen on objects with many sharp edges or where high contrast bump or normal maps are applied.

### Reflection Samples

Controls the amount of blur of the surface reflections. Some amount of blur is required in order to have the Dispersion amount affect the surface. The number of samples used also greatly affects the quality of blurring that takes place.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

### Normal

Normal maps connected to this input will replace the normals that this shader applies shading to in an individually exclusive manner from other shading models. This means that more than one set of normals can be utilized by any shaders exclusively or shared between any of the various shaders and/or the global normals for an individual surface.

The per-surface global normals are defined by either the interpolated geometry itself or by whatever is connected to the Normal input on the Surface destination node.

By using a different set of normals either from different normal maps or from any of various nodes that may offer normal outputs themselves, it becomes possible to create truly layered surface shading and texturing.

Can receive input from other vector type outputs that define surface normal information. Usually these outputs contain the word "Normal" in the output label text.

### Bump

Specifies a vector direction for modifying the surface normal. This is surface normal direction information and affects the way the surface is shaded. Care should be taken when connecting to this input. Connecting dissimilar types or non directional vectors may cause the surface to shade wrongly.

Can receive input from other nodes in the network.

### Bump Height

Specifies the bump height or "amplitude" of the Bump directional vectors.

Using Alpha or luminance (connecting Color) values from other textures in the network along with their respective bumps can have the effect of blending the bump textures together.

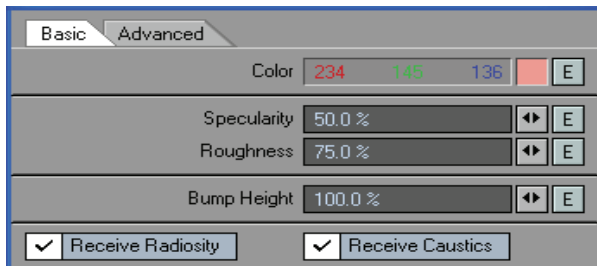
Capable of receiving patterns or numerical values from other nodes in the network, this value can additionally be specified by user input by entering values into the Edit Panel of this node.

Since Bump Amplitude is a shading feature and not a displacement coordinate system value it is specified as a percentage, specifically as a percentage of the texture value.





## Edit Panel



### Basic Tab

#### Receive Radiosity/ Receive Caustics

Here the Radiosity and Caustics check boxes are not offered as node connections in the Workspace area and are features exclusive to the Edit Panel of the node.

When checked this shading model will evaluate shading information for the surface channel it is being applied including any Radiosity or Caustics computations as defined by those system attributes applied to the scene.

Unchecked, radiosity and/or caustics calculations will not be evaluated by this shading model as it is applied to the object surface being edited.

This can be used to omit both entire surfaces or per channel surface attributes from radiosity or caustics calculations thus offering more diversity in the rendered output as well as reducing the time needed to calculate these surfaces in scenes where either radiosity or caustics are used.

### Advanced Tab

#### Mode

#### Reflection Image

Mode and Image are features exclusive to this nodes Edit Panel . Please see the explanation for these items in the chapter that covers texturing with the Surface Editor.

#### Image Seam Angle

Angle or "Image Seam Angle" as it is designated in the Edit Panel for this node allows the user to rotate the seam line of a wrapped image around the Axis that has been selected for the projection method.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

## Samples

Can receive numerical inputs from other nodes in the network. Additionally, controls offered in the Edit Panel for this node may be used to specify this value.

Sampling is the number of directions the shading model examines the surface from in order to calculate the shading value of each spot on the surface under evaluation.

These readings or "Samples" are taken from positions on a hemispherical lattice and it is the number of longitudinal and latitudinal sections of this hemisphere that you are defining when you select one of the Sample preset values from the Samples pull down menu. It is for this reason that there are two numbers in each of the possible Sample selections.

It's quite easy to understand if you imagine one half of an unsmoothed (faceted) default ball object in modeler. The first number defines the number of longitudinal strips and the second number defines the number of latitudinal sections where each facet (or "face") formed by their intersection defines one of the directions from which the spot is evaluated. You could visualize a sample size of 1x4 for example, as a bottomless pyramid centered over the evaluation spot protruding straight out from the surface normal - 1 segment high and containing 4 sides.

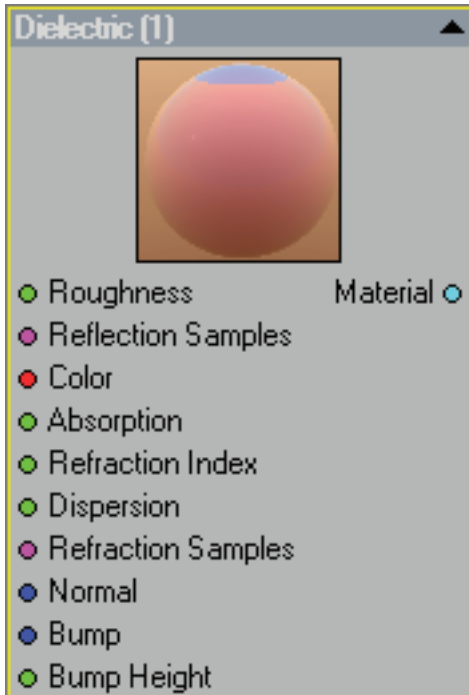
Higher values will sample the shading from more directions and produce smoother more accurate shading.

Incoming values higher than 15 will be set to 15. Incoming values lower than 0 will be set to 0.



## Dielectric

A physically accurate node for simulating glass-like and liquid materials. Use this state-of-the-art node when you want to create realistic glass. Dielectric uses Beer's Law, which is about energy absorption, occurring when light passes through a surface. The more light that is absorbed by the material, the darker it will look.



## Inputs

### Roughness

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Specifies the amount of roughness that will be applied. Roughness is the result of microfacet evaluation in lighting conditions. Higher values produce rougher surface properties while lower values produce smoother looking surfaces. This kind of roughness is more easily seen on objects with many sharp edges or where high contrast bump or normal maps are applied.

### Reflection Samples

Controls the amount of blur of the surface reflections. Some amount of blur is required in order to have the Dispersion amount affect the surface. The number of samples used also greatly affects the quality of blurring that takes place.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

### Color

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

## Absorption

Controls the amount of light dispersion. This input can receive patterns or numerical values from other nodes in the network. This value can additionally be specified via user input by entering values into the Edit Panel for this node.

Dispersion is a phenomenon that causes the separation of a wave into spectral components with different wavelengths, due to a dependence of the waves' speed on its wavelength.

In simpler terms dispersion occurs in materials in which the Index Of Refraction (IOR) is not constant, but rather varies depending on the wavelength of incoming light. White light splits into its monochromatic components, where each wavelength beam has a different IOR. The classic example is a beam of white light entering a prism of some sort and projecting a rainbow colored pattern out the other side.

Dispersion as used in this shading model simulates real light dispersion. The higher the value the greater the amount of dispersion that will be simulated and applied.

Delta is an energy conserving material. What this means is that it has realistic properties like when specularity is 100% the diffuse in turn will go down to 0%.

## Refraction Samples

Index Of Refraction or IOR, is a charted index of the different amounts of refraction associated with real world materials.

Refraction is the bending of light as it passes through transparent materials or substances of different density.

Dense substances bend light more than materials that are not dense.

See elsewhere in this manual for a list of many real world transparent materials and their corresponding IOR values.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

## Normal

Normal maps connected to this input will replace the normals that this shader applies shading to in an individually exclusive manner from other shading models. This means that more than one set of normals can be utilized by any shaders exclusively or shared between any of the various shaders and/or the global normals for an individual surface.

The per-surface global normals are defined by either the interpolated geometry itself or by whatever is connected to the Normal input on the Surface destination node.

By using a different set of normals either from different normal maps or from any of various nodes that may offer normal outputs themselves, it becomes possible to create truly layered surface shading and texturing.

Can receive input from other vector type outputs that define surface normal information. Usually these outputs contain the word "Normal" in the output label text.



## Bump

Specifies a vector direction for modifying the surface normal. This is surface normal direction information and affects the way the surface is shaded. Care should be taken when connecting to this input. Connecting dissimilar types or non directional vectors may cause the surface to shade wrongly.

Can receive input from other nodes in the network.

## Bump Height

Specifies the bump height or “amplitude” of the Bump directional vectors.

Using Alpha or luminance (connecting Color) values from other textures in the network along with their respective bumps can have the effect of blending the bump textures together.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

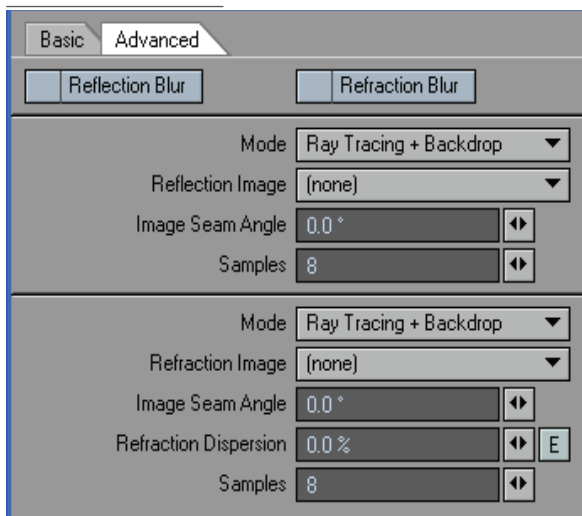
Since Bump Amplitude is a shading feature and not a displacement coordinate system value it is specified as a percentage. Specifically as a percentage of the texture value.

## Edit Panel

### Basic Tab



### Advanced Tab



## Mode

## Reflection Image

Mode and Image are features exclusive to this nodes Edit Panel . Please see the explanation for these items in the chapter that covers texturing with the Surface Editor.

## Image Seam Angle

Angle or “Image Seam Angle” as it is designated in the Edit Panel for this node allows the user to rotate the seam line of a wrapped image around the Axis that has been selected for the projection method.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

## Samples

Can receive numerical inputs from other nodes in the network. Additionally, controls offered in the Edit Panel for this node may be used to specify this value.

Sampling is the number of directions the shading model examines the surface from in order to calculate the shading value of each spot on the surface under evaluation.

These readings or “Samples” are taken from positions on a hemispherical lattice and it is the number of longitudinal and latitudinal sections of this hemisphere that you are defining when you select one of the Sample preset values from the Samples pull down menu. It is for this reason that there are two numbers in each of the possible Sample selections.

It’s quite easy to understand if you imagine one half of an unsmoothed (faceted) default ball object in modeler. The first number defines the number of longitudinal strips and the second number defines the number of latitudinal sections where each facet (or “face”) formed by their intersection defines one of the directions from which the spot is evaluated. You could visualize a sample size of 1x4 for example, as a bottomless pyramid centered over the evaluation spot protruding straight out from the surface normal - 1 segment high and containing 4 sides.

Higher values will sample the shading from more directions and produce smoother more accurate shading.

Incoming values higher than 15 will be set to 15. Incoming values lower than 0 will be set to 0.

## Image Seam Angle

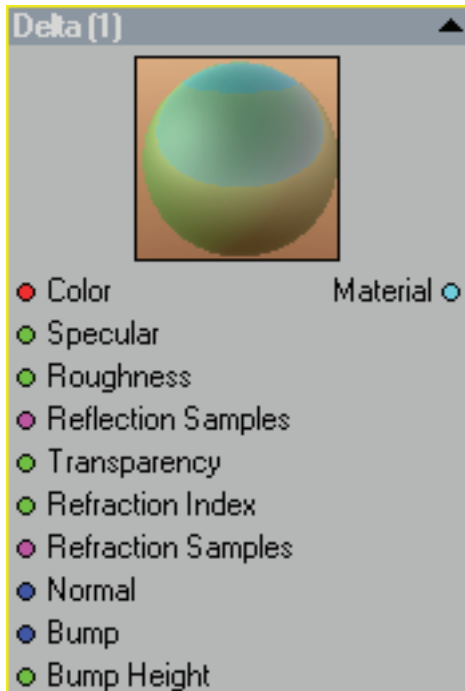
Angle or “Image Seam Angle” as it is designated in the Edit Panel for this node allows the user to rotate the seam line of a wrapped image around the Axis that has been selected for the projection method.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.



## Delta

Delta is an energy conserving material. What this means is that it has realistic properties like when specularity is 100% the diffuse in turn will go down to 0%.



## Inputs

### Color

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

### Specularity

Defines the amount of specularity that is applied to the surface by this node.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

### Roughness

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Specifies the amount of roughness that will be applied. Roughness is the result of microfacet evaluation in lighting conditions. Higher values produce rougher surface properties while lower values produce smoother looking surfaces. This kind of roughness is more easily seen on objects with many sharp edges or where high contrast bump or normal maps are applied.

## Samples

Can receive numerical inputs from other nodes in the network. Additionally, controls offered in the Edit Panel for this node may be used to specify this value.

Sampling is the number of directions the shading model examines the surface from in order to calculate the shading value of each spot on the surface under evaluation.

These readings or "Samples" are taken from positions on a hemispherical lattice and it is the number of longitudinal and latitudinal sections of this hemisphere that you are defining when you select one of the Sample preset values from the Samples pull down menu. It is for this reason that there are two numbers in each of the possible Sample selections.

It's quite easy to understand if you imagine one half of an unsmoothed (faceted) default ball object in modeler. The first number defines the number of longitudinal strips and the second number defines the number of latitudinal sections where each facet (or "face") formed by their intersection defines one of the directions from which the spot is evaluated. You could visualize a sample size of 1x4 for example, as a bottomless pyramid centered over the evaluation spot protruding straight out from the surface normal - 1 segment high and containing 4 sides.

Higher values will sample the shading from more directions and produce smoother more accurate shading.

## Transparency

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Specifies the amount of transparency applied to the surface.

## Refraction Index

Index Of Refraction or IOR, is a charted index of the different amounts of refraction associated with real world materials.

Refraction is the bending of light as it passes through transparent materials or substances of different density.

Dense substances bend light more than materials that are not dense.

See elsewhere in this manual for a list of many real world transparent materials and their corresponding IOR values.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.



## Normal

Normal maps connected to this input will replace the normals that this shader applies shading to in an individually exclusive manner from other shading models. This means that more than one set of normals can be utilized by any shaders exclusively or shared between any of the various shaders and/or the global normals for an individual surface.

The per-surface global normals are defined by either the interpolated geometry itself or by whatever is connected to the Normal input on the Surface destination node.

By using a different set of normals either from different normal maps or from any of various nodes that may offer normal outputs themselves, it becomes possible to create truly layered surface shading and texturing.

Can receive input from other vector type outputs that define surface normal information. Usually these outputs contain the word "Normal" in the output label text.

## Bump

Specifies a vector direction for modifying the surface normal. This is surface normal direction information and affects the way the surface is shaded. Care should be taken when connecting to this input. Connecting dissimilar types or non directional vectors may cause the surface to shade wrongly.

Can receive input from other nodes in the network.

## Bump Height

Specifies the bump height or "amplitude" of the Bump directional vectors.

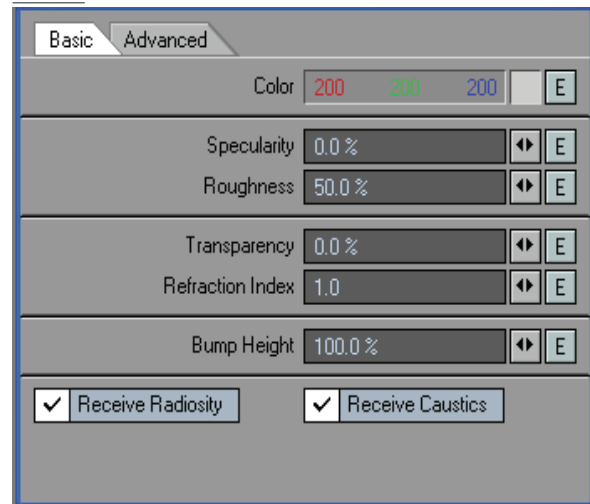
Using Alpha or luminance (connecting Color) values from other textures in the network along with their respective bumps can have the effect of blending the bump textures together.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Since Bump Amplitude is a shading feature and not a displacement coordinate system value it is specified as a percentage. Specifically as a percentage of the texture value.

## Edit Panel

### Basic Tab



### Receive Radiosity

### Receive Caustics

Here the Radiosity and Caustics check boxes are not offered as node connections in the Workspace area and are features exclusive to the Edit Panel of the node.

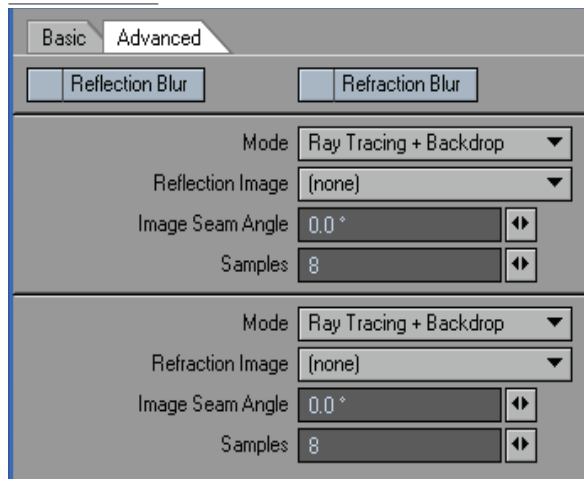
When checked this shading model will evaluate shading information for the surface channel it is being applied including any Radiosity or Caustics computations as defined by those system attributes applied to the scene.

Unchecked, radiosity and/or caustics calculations will not be evaluated by this shading model as it is applied to the object surface being edited.

This can be used to omit both entire surfaces or per channel surface attributes from radiosity or caustics calculations thus offering more diversity in the rendered output as well as reducing the time needed to calculate these surfaces in scenes where either radiosity or caustics are used.



## Advanced Tab



### Reflection Blur

Controls the amount of blur of the surface reflections. Some amount of blur is required in order to have the Dispersion amount affect the surface. The number of samples used also greatly affects the quality of blurring that takes place.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

### Mode and Reflection Image

Mode and Image are features exclusive to this nodes Edit Panel . Please see the explanation for these items in the chapter that covers texturing with the Surface Editor.

### Image Seam Angle

Angle or "Image Seam Angle" as it is designated in the Edit Panel for this node allows the user to rotate the seam line of a wrapped image around the Axis that has been selected for the projection method.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

For a table of the possible sampling matrices and the numerical values associated with each, see the end of this document.

## Fast Skin

Fast skin is a skin shader with improved subsurface scattering and less noise.







## Edit Panel

Basic		Advanced	
Diffuse Color	097 107 107		E
Diffuse	30.0 %	◀▶	E
Respect Bump	0.0 %	◀▶	E
Specular Color	255 255 255		E
Specularity	80.0 %	◀▶	E
Glossiness	65.0 %	◀▶	E
Fresnel	80.0 %	◀▶	E
Refraction Index	1.37	◀▶	E
Epidermis Visibility	90.0 %	◀▶	E
Epidermis Back Scatter	100.0 %	◀▶	E
Epidermis Color	224 199 153		E
Epidermis Distance	2 mm	◀▶	E
Epidermis Gamma	1.0	◀▶	E
Subdermis Visibility	90.0 %	◀▶	E
Subdermis Back Scatter	100.0 %	◀▶	E
Subdermis Color	201 088 069		E
Subdermis Distance	5 mm	◀▶	E
Subdermis Gamma	1.0	◀▶	E
Quality	100.0 %	◀▶	E
Bump Height	100.0 %	◀▶	E
<input checked="" type="checkbox"/> Receive Radiosity		<input checked="" type="checkbox"/> Receive Caustics	

## Diffuse Color

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color used for the diffuse translucent channel.

## Diffuse

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Specifies the amount of Diffuse reflectivity that will be applied.

## Diffuse Roughness

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Specifies the amount of roughness that will be applied. Roughness is the result of microfacet evaluation in lighting conditions. Higher values produce rougher surface properties while lower values produce smoother looking surfaces. This kind of roughness is more easily seen on objects with many sharp edges or where high contrast bump or normal maps are applied.

## Specular Color

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

## Specularity

Defines the amount of specularity that is applied to the surface by this node.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

## Glossiness

Defines the *amount* of glossiness that is applied to the surface by this node.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

## Fresnel

Defines the amount light reflection on a surface.

## Refraction Index

Index Of Refraction or IOR, is a charted index of the different amounts of refraction associated with real world materials.

Refraction is the bending of light as it passes through transparent materials or substances of different density.

Dense substances bend light more than materials that are not dense.

See elsewhere in this manual for a list of many real world transparent materials and their corresponding IOR values.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

## Epidermis Visibility

Determines the visibility of the outer-layer of the skin shader.

## Epidermis Color

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color used for the outer-layer of skin.

## Epidermis Distance

The depth into how far light is absorbed into a surface.

## Epidermis Gamma

This function provides a standard gamma correction function. Gamma correction is a function that adjusts the brightness and contrast simultaneously.



## Make Material

Used to make a material from the older shader components.



### Inputs

#### Diffuse

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Specifies the amount of Diffuse reflectivity that will be applied.

#### Specular

Defines the amount of specularity that is applied to the surface by this node.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

#### Reflection

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Defines the amount of reflection that is applied to the surface by this node.

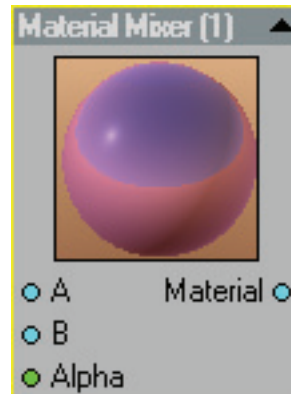
#### Refraction

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Defines the amount of refraction that is applied to the surface by this node.

## Material Mixer

Material Mixer blends materials together, the strength of the blend determined by the alpha channel.



### Inputs

#### A/B

The two Materials which will be blended.

#### Alpha

Using the alpha information from another mode, this input determines how the materials are mixed.



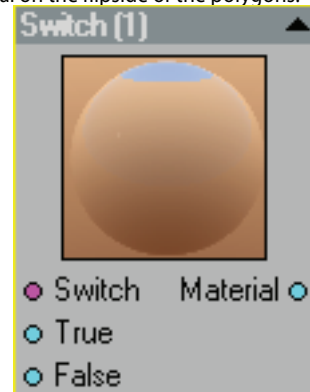
## Standard

Standard is a replication of the built-in shading model. Build custom materials with this node.



## Switch

Switch switches between the materials using the integer input. It is mainly meant for air polygons, where you need to have different material on the flipside of the polygons.



## Switch

This input determines where the switch will be made. Typically you will use the Spot Info node and the Polygon Side output for best use of this input.

## True

Used for the positive side of the polygons.

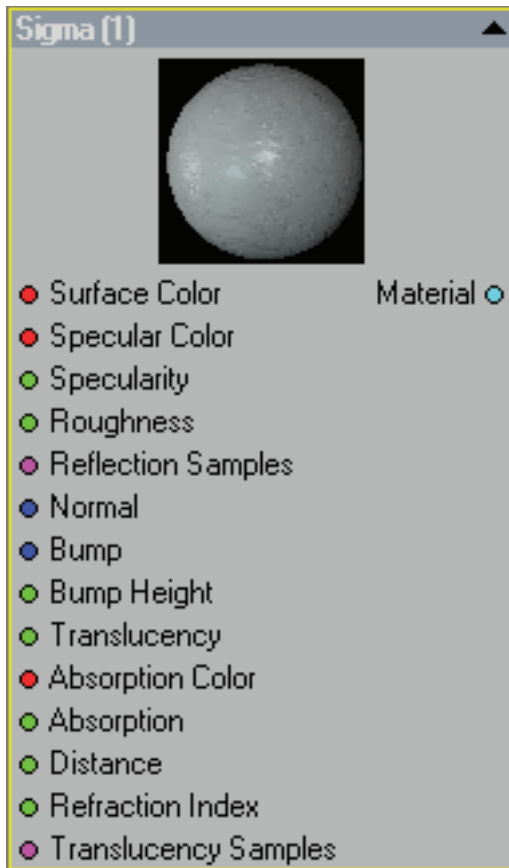
## False

Used for the negative side of the polygons.



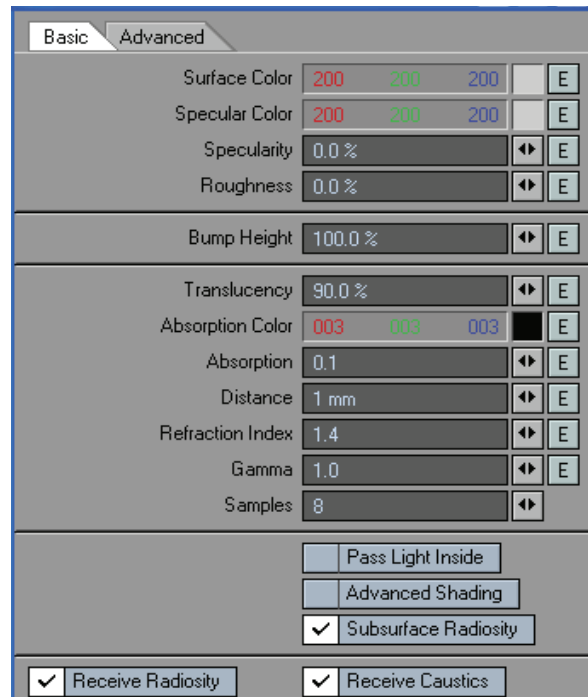
## Sigma

A material node which uses subsurface scattering, which occurs when light partially passes through a surface, and is scattered within the object, then exits through a different location. Subsurface scattering is an important and recent addition 3D computer graphics for the realistic rendering of such materials as marble, skin, and milk and other real-world semitransparent substances.



## Edit Panel

### Basic Tab



### Surface Color

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

### Specularity Color

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

### Specularity

Defines the amount of specularity that is applied to the surface by this node. Also has the affect of reducing the diffuse, so when Specularity is set to 100%, diffuse will be 0%.

### Roughness

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Specifies the amount of roughness that will be applied. Roughness is the result of microfacet evaluation in lighting conditions. Higher values produce rougher surface properties while lower values produce smoother looking surfaces. This kind of roughness is more easily seen on objects with many sharp edges or where high contrast bump or normal maps are applied.



## Bump Height

Specifies the bump height or “amplitude” of the Bump directional vectors.

Using Alpha or luminance (connecting Color) values from other textures in the network along with their respective bumps can have the effect of blending the bump textures together.

Capable of receiving patterns or numerical values from other nodes in the network, this value can additionally be specified by user input by entering values into the Edit Panel of this node.

Since Bump Amplitude is a shading feature and not a displacement coordinate system value it is specified as a percentage, specifically as a percentage of the texture value.

## Translucency

Translucency is the material quality of allowing light to pass diffusely through semitransparent substances.

Since translucency is in consideration of light rays in specific, it is considered and classified as a diffuse shading model.

Translucent shaded surfaces will not reveal the surface colors and properties of other object surfaces that exist on the other side of the translucent object away from the observer. Lights however will show through translucent surfaces and are needed in order for translucency shaders to affect the surface at all.

## Absorption Color

The color channel which is absorbed. Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

## Absorption

The amount of light that is absorbed.

## Distance

The depth into how far light is absorbed into a surface

## Refraction Index

Index Of Refraction or IOR, is a charted index of the different amounts of refraction associated with real world materials.

Refraction is the bending of light as it passes through transparent materials or substances of different density.

Dense substances bend light more than materials that are not dense.

See Suracing & Rendering manual for a list of many real world transparent materials and their corresponding IOR values.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

## Samples

Can receive numerical inputs from other nodes in the network. Additionally, controls offered in the Edit Panel for this node may be used to specify this value.

Sampling is the number of directions the shading model examines the surface from in order to calculate the shading value of each spot on the surface under evaluation.

These readings or “Samples” are taken from positions on a hemispherical lattice and it is the number of longitudinal and latitudinal sections of this hemisphere that you are defining when you select one of the Sample preset values from the Samples pull down menu. It is for this reason that there are two numbers in each of the possible Sample selections.

It's quite easy to understand if you imagine one half of an unsmoothed (faceted) default ball object in modeler. The first number defines the number of longitudinal strips and the second number defines the number of latitudinal sections where each facet (or “face”) formed by their intersection defines one of the directions from which the spot is evaluated. You could visualize a sample size of 1x4 for example, as a bottomless pyramid centered over the evaluation spot protruding straight out from the surface normal - 1 segment high and containing 4 sides.

Higher values will sample the shading from more directions and produce smoother more accurate shading.

Incoming values higher than 15 will be set to 15. Incoming values lower than 0 will be set to 0.

## Pass Light Inside

If on, light is passed inside the surface. Which means all surfaces inside the Sigma surface receive illumination. If it is off, no light passes inside the surface.

## Advanced Shading

If this is on the SSS rays will hit any other surfaces inside the SSS surface, otherwise the rays will only hit the SSS surface.

## Subsurface Radiosity

When checked, Sigma will calculate the radiosity for the subsurface of the surface.

## Receive Radiosity/ Receive Caustics

Here the Radiosity and Caustics check boxes are not offered as node connections in the Workspace area and are features exclusive to the Edit Panel of the node.

When checked this shading model will evaluate shading information for the surface channel it is being applied including any Radiosity or Caustics computations as defined by those system attributes applied to the scene.

Unchecked, radiosity and/or caustics calculations will not be evaluated by this shading model as it is applied to the object surface being edited.

This can be used to omit both entire surfaces or per channel surface attributes from radiosity or caustics calculations thus offering more diversity in the rendered output as well as reducing the time needed to calculate these surfaces in scenes where either radiosity or caustics are used.



## Advanced Tab

### Reflection Blur

Controls the amount of blur of the surface reflections. Some amount of blur is required in order to have the Dispersion amount affect the surface. The number of samples used also greatly affects the quality of blurring that takes place.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

### Mode/ Reflection Image

Mode and Image are features exclusive to this nodes Edit Panel . Please see the explanation for these items in the chapter that covers texturing with the Surface Editor.

### Image Seam Angle

Angle or "Image Seam Angle" as it is designated in the Edit Panel for this node allows the user to rotate the seam line of a wrapped image around the Axis that has been selected for the projection method.

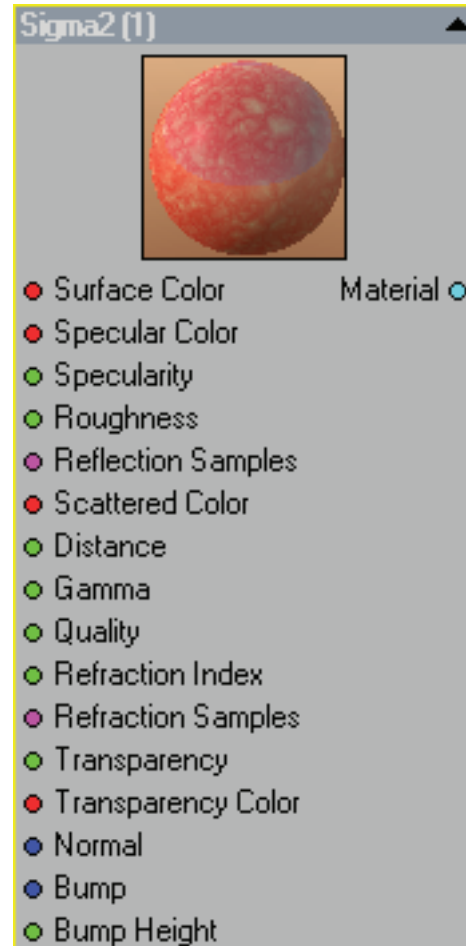
Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

### Occlusion

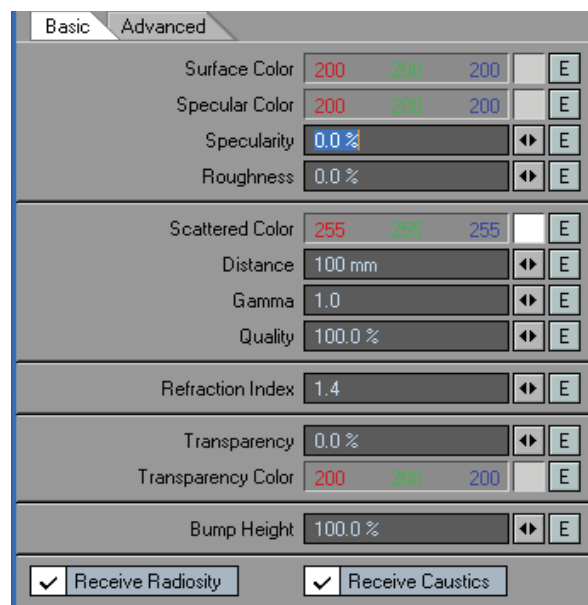
Occlusion or "ambient occlusion" is a shading method used to help add realism by taking into account attenuation of light caused by obstructive geometry.

## Sigma2

A material node which uses subsurface scattering, which occurs when light partially passes through a surface, and is scattered within the object, then exits through a different location. Subsurface scattering is an important and recent addition 3D computer graphics for the realistic rendering of such materials as marble, skin, and milk and other real-world semitransparent substances.



## Basic Tab







## Surface Color

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

## Specularity Color

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

## Specularity

Defines the amount of specularity that is applied to the surface by this node. Also has the affect of reducing the diffuse, so when Specularity is set to 100%, diffuse will be 0%.

## Roughness

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Specifies the amount of roughness that will be applied. Roughness is the result of microfacet evaluation in lighting conditions. Higher values produce rougher surface properties while lower values produce smoother looking surfaces. This kind of roughness is more easily seen on objects with many sharp edges or where high contrast bump or normal maps are applied.

## Bump

Specifies a vector direction for modifying the surface normal. This is surface normal direction information and affects the way the surface is shaded. Care should be taken when connecting to this input. Connecting dissimilar types or non directional vectors may cause the surface to shade wrongly.

Can receive input from other nodes in the network only.

## Bump Height

Specifies the bump height or "amplitude" of the Bump directional vectors.

Using Alpha or luminance (connecting Color) values from other textures in the network along with their respective bumps can have the effect of blending the bump textures together.

Capable of receiving patterns or numerical values from other nodes in the network, this value can additionally be specified by user input by entering values into the Edit Panel of this node.

Since Bump Amplitude is a shading feature and not a displacement coordinate system value it is specified as a percentage, specifically as a percentage of the texture value.

## Samples

Can receive numerical inputs from other nodes in the network. Additionally, controls offered in the Edit Panel for this node may be used to specify this value.

Sampling is the number of directions the shading model examines the surface from in order to calculate the shading value of each spot on the surface under evaluation.

These readings or "Samples" are taken from positions on a hemispherical lattice and it is the number of longitudinal and

latitudinal sections of this hemisphere that you are defining when you select one of the Sample preset values from the Samples pull down menu. It is for this reason that there are two numbers in each of the possible Sample selections.

It's quite easy to understand if you imagine one half of an unsmoothed (faceted) default ball object in modeler. The first number defines the number of longitudinal strips and the second number defines the number of latitudinal sections where each facet (or "face") formed by their intersection defines one of the directions from which the spot is evaluated. You could visualize a sample size of 1x4 for example, as a bottomless pyramid centered over the evaluation spot protruding straight out from the surface normal - 1 segment high and containing 4 sides.

Higher values will sample the shading from more directions and produce smoother more accurate shading.

Incoming values higher than 15 will be set to 15. Incoming values lower than 0 will be set to 0.

## Scattered Color

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

## Distance

The depth into how far light is absorbed into a surface

## Quality

Controls the quality of the subsurface scattering.

## Refraction Index

Index Of Refraction or IOR, is a charted index of the different amounts of refraction associated with real world materials.

Refraction is the bending of light as it passes through transparent materials or substances of different density.

Dense substances bend light more than materials that are not dense.

See Surfacing & Rendering manual for a list of many real world transparent materials and their corresponding IOR values.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.



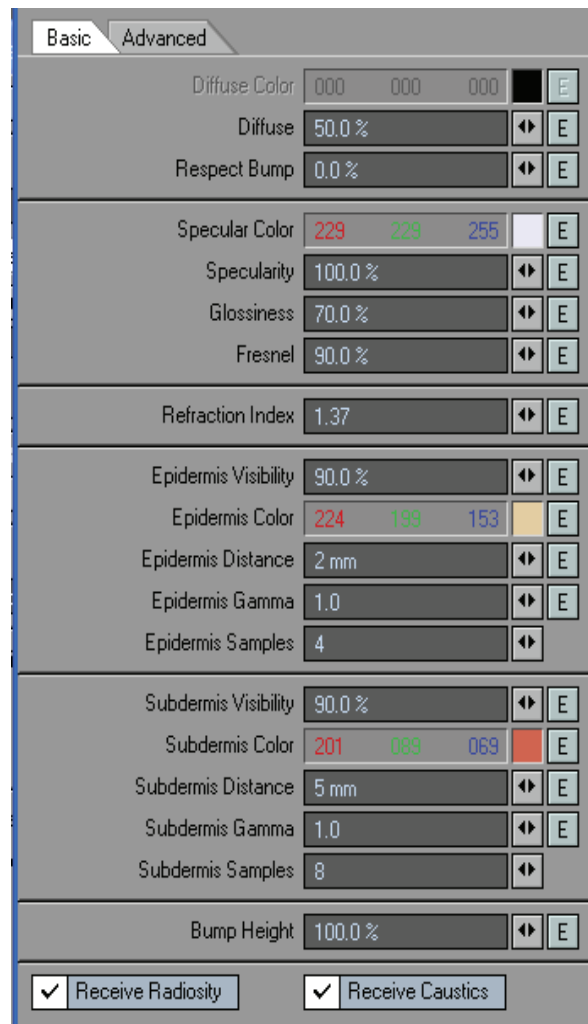
## Simple Skin

Simple Skin, as the name implies, is a physically accurate skin shader with subsurface scattering. Subsurface scattering occurs when light partially passes through a surface, and is scattered within the object, then exits through a different location.



## Edit Panel

### Basic Tab



## Diffuse Color

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color used for the diffuse translucent channel.

## Diffuse

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Specifies the amount of Diffuse reflectivity that will be applied.



## Diffuse Roughness

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Specifies the amount of roughness that will be applied. Roughness is the result of microfacet evaluation in lighting conditions. Higher values produce rougher surface properties while lower values produce smoother looking surfaces. This kind of roughness is more easily seen on objects with many sharp edges or where high contrast bump or normal maps are applied.

## Specular Color

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

## Specularity

Defines the amount of specularity that is applied to the surface by this node.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

## Glossiness

Defines the *amount* of glossiness that is applied to the surface by this node.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

## Fresnel

Defines the amount light reflection on a surface.

## Refraction Index

Index Of Refraction or IOR, is a charted index of the different amounts of refraction associated with real world materials.

Refraction is the bending of light as it passes through transparent materials or substances of different density.

Dense substances bend light more than materials that are not dense.

See elsewhere in this manual for a list of many real world transparent materials and their corresponding IOR values.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

## Epidermis Visibility

Determines the visibility of the outer-layer of the skin shader.

## Epidermis Color

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color used for the outer-layer of skin.

## Epidermis Distance

The depth into how far light is absorbed into a surface.

## Epidermis Gamma

This function provides a standard gamma correction function. Gamma correction is a function that adjusts the brightness and contrast simultaneously.

## Epidermis Samples

Can receive numerical inputs from other nodes in the network. Additionally, controls offered in the Edit Panel for this node may be used to specify this value.

Sampling is the number of directions the shading model examines the surface from in order to calculate the shading value of each spot on the surface under evaluation.

These readings or "Samples" are taken from positions on a hemispherical lattice and it is the number of longitudinal and latitudinal sections of this hemisphere that you are defining when you select one of the Sample preset values from the Samples pull down menu. It is for this reason that there are two numbers in each of the possible Sample selections.

It's quite easy to understand if you imagine one half of an unsmoothed (faceted) default ball object in modeler. The first number defines the number of longitudinal strips and the second number defines the number of latitudinal sections where each facet (or "face") formed by their intersection defines one of the directions from which the spot is evaluated. You could visualize a sample size of 1x4 for example, as a bottomless pyramid centered over the evaluation spot protruding straight out from the surface normal - 1 segment high and containing 4 sides.

Higher values will sample the shading from more directions and produce smoother more accurate shading.

Incoming values higher than 15 will be set to 15. Incoming values lower than 0 will be set to 0.



## Subdermis Visibility

Determines the visibility of the sub-layer of the skin shader.

## Subdermis Color

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color used for the sub-layer of skin.

## Subdermis Distance

The depth into how far light is absorbed into a surface

## Subdermis Gamma

This function provides a standard gamma correction function. Gamma correction is a function that adjusts the brightness and contrast simultaneously.

## Subdermis Samples

Can receive numerical inputs from other nodes in the network. Additionally, controls offered in the Edit Panel for this node may be used to specify this value.

Sampling is the number of directions the shading model examines the surface from in order to calculate the shading value of each spot on the surface under evaluation.

These readings or "Samples" are taken from positions on a hemispherical lattice and it is the number of longitudinal and latitudinal sections of this hemisphere that you are defining when you select one of the Sample preset values from the Samples pull down menu. It is for this reason that there are two numbers in each of the possible Sample selections.

It's quite easy to understand if you imagine one half of an unsmoothed (faceted) default ball object in modeler. The first number defines the number of longitudinal strips and the second number defines the number of latitudinal sections where each facet (or "face") formed by their intersection defines one of the directions from which the spot is evaluated. You could visualize a sample size of 1x4 for example, as a bottomless pyramid centered over the evaluation spot protruding straight out from the surface normal - 1 segment high and containing 4 sides.

Higher values will sample the shading from more directions and produce smoother more accurate shading.

Incoming values higher than 15 will be set to 15. Incoming values lower than 0 will be set to 0.

## Bump Height

Specifies the bump height or "amplitude" of the Bump directional vectors.

Using Alpha or luminance (connecting Color) values from other textures in the network along with their respective bumps can have the effect of blending the bump textures together.

Capable of receiving patterns or numerical values from other nodes in the network, this value can additionally be specified by user input by entering values into the Edit Panel of this node.

Since Bump Amplitude is a shading feature and not a displacement coordinate system value it is specified as a percentage, specifically as a percentage of the texture value.

## Receive Radiosity/ Receive Caustics

Here the Radiosity and Caustics check boxes are not offered as node connections in the Workspace area and are features exclusive to the Edit Panel of the node.

When checked this shading model will evaluate shading information for the surface channel it is being applied including any Radiosity or Caustics computations as defined by those system attributes applied to the scene.

Unchecked, radiosity and/or caustics calculations will not be evaluated by this shading model as it is applied to the object surface being edited.

This can be used to omit both entire surfaces or per channel surface attributes from radiosity or caustics calculations thus offering more diversity in the rendered output as well as reducing the time needed to calculate these surfaces in scenes where either radiosity or caustics are used.

## Advanced Tab

### Reflection Blur

Controls the amount of blur of the surface reflections. Some amount of blur is required in order to have the Dispersion amount affect the surface. The number of samples used also greatly affects the quality of blurring that takes place.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

## Mode/ Reflection Image

Mode and Image are features exclusive to this nodes Edit Panel. Please see the explanation for these items in the chapter that covers texturing with the Surface Editor.

## Image Seam Angle

Angle or "Image Seam Angle" as it is designated in the Edit Panel for this node allows the user to rotate the seam line of a wrapped image around the Axis that has been selected for the projection method.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.



## Math (Scalar)

### Abs

#### Purpose

Perform the Abs function on scalar values and output the results.

#### Description

In mathematics, the modulus or “absolute value” (abbreviated Abs) of a real number is its numerical value without regard to its sign. So, for example, 5 is the absolute value of both 5 and  $-5$  and 3.14159 is the absolute value of both  $-3.14159$  and 3.14159.

In a general frame of reference you can think of this nodes function as changing negative values into positive values or in other words inverting only the numbers that exist on the negative side of a number line. The absolute value of a number is always either positive or zero, never negative.



#### Inputs

In (type: scalar):

Outputs from other nodes that are connected here will have the Abs function applied to them.

If the incoming value is not a scalar type value it will automatically be converted into it's scalar equivalent for this operation.

#### Outputs

Out (type: scalar):

The Out result output will supply the Abs value of Input for connecting to inputs parameters of other nodes.

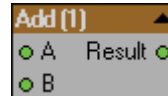
#### Edit Panel

None

## Add

#### Purpose

To add the value in B to the value in A (both scalar) and return their sum as the resulting output.



#### Description

The Add node adds the incoming or user specified value “B” to the incoming or user specified value “A” and supplies their sum as the “Result” output.

It is important to note the order of the values in all the math nodes as inverse operations do not always produce the same result. Addition and multiplication are exceptions but if you keep this rule throughout all of the math nodes you will run into fewer problems.

#### Inputs

A (type: scalar):

Outputs from other nodes that are connected here will have B added to it's value.

B (type: scalar):

Outputs from other nodes that are connected here specify the value that the value in A is added to.

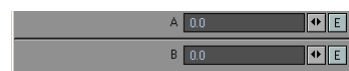
Both A and B can be specified by the user by double clicking on the node in the work area and entering a value manually. Incoming values from other nodes connected to the A or B inputs in the work area will override any user specified values respectively.

#### Outputs

Result (type: scalar):

The result of the add operation can be connected to the input parameters of other nodes

#### Edit Panel





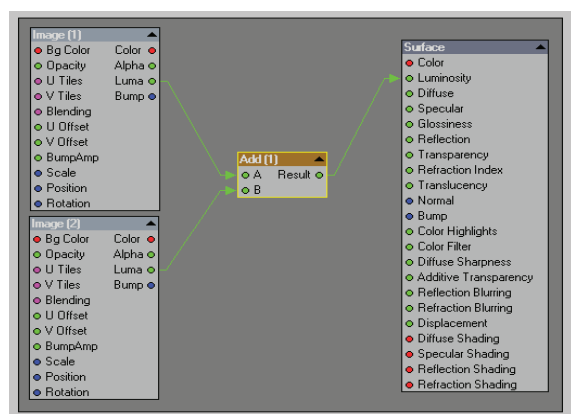
## Usage Example

Grayscale images are one kind of scalar data item. They contain only one channel. This is opposed to color images which contain 3 planes of data R, G, and B per pixel values.

In this simple example we will use the Add node to add two image Luma (grey information) channels together and view the results on the luminosity channel of a named surface. The pixel value for the graphic of the *A* area in the first image is 120 and the *B* area of the second image is 100. When added together (where they intersect) the grey value should be 220. Since both backgrounds are black which has a pixel value of zero the BG area should remain black as  $0+0=0$ . Where the *A* area overlaps the black BG of the *B* image it should retain it's original grey value as  $120+0=120$  and so on.



Let's take a quick look at the work area of the Node editor in order to see how we achieved this result using the Add node.



In order to duplicate this example first the Diffuse in the Surface Editor Panel should be set to 0% so as not to mix the diffuse shading with the result of our node construction. After opening the Node Edit panel by clicking on the Edit Nodes button we are ready to add the Image nodes and select the relative images. (See Textures > Image for information on how to use the Image node.) Once the images are loaded and scaled to the object surface the Add node is used to add the two grey scale images together by connecting the inputs and outputs as shown above.

The Add node can be used on many data item types other than grey scale images.

## BoxStep

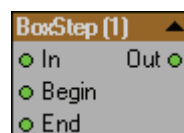
### Purpose

The BoxStep Math function provides a standard box step function that can be applied to any scalar.

### Description

A box step is a ramped step function. Two limits are set by the Begin and End inputs. If the input is less than the Begin input, then the output will be 0.0. Conversely, if the Begin input is greater than the End input, the output will be 1.0. If the input is between Begin and End, then the output will vary between 0.0 and 1.0 proportionally.

Effectively, the Box Step clamps between the Begin and End values and normalizes the output between 0.0 and 1.0.



### Inputs

In (type: scalar):

The In input specifies the input to the box step function.

Begin (type: scalar):

The Begin input specifies the start of the box step ramp. If In is below this value, then the output is 0.0. If In is between the Begin and End inputs, then the output will vary between 0.0 and 1.0, proportionally.

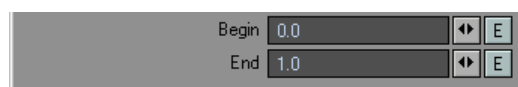
End (type: scalar):

The End input specifies the end of the box step ramp. If In is above this value, then the output is 1.0. If In is between the Begin and End inputs, then the output will vary between 0.0 and 1.0, proportionally.

Out (type: scalar):

The Out output will vary between 0.0 and 1.0 as the input varies between the values of Begin and End.

### Edit Panel







## Ceil

### Purpose

To supply other nodes with the ceiling level value from any of various data items input as scalar type values.



### Description

Ceil is the abbreviation for “ceiling”. This node receives scalar data from an arbitrary data item such as the luminance values of an image, the Alpha channel of a procedural or the output from another math node. It evaluates the incoming value and rounds that value up (hence the name “ceiling”) to the nearest whole number which is then output as the result.

This can be very useful for extracting stepped data from smooth gradations among many other things. See the I/O graph below to get an idea of what is going on under the hood:

Input:	0.2	0.6	1.0	1.4	1.8	2.2	2.6	3.0
Result:	1	1	1	2	2	3	3	3

### Inputs

In (type: scalar):

Outputs from other nodes need to be connected to the Ceil input in order for the operation to be applied on the incoming data.

### Outputs

Out (type: scalar):

The results of the Ceiling operation can be connected to other nodes for further processing or directly connected to the Surface Destination node.

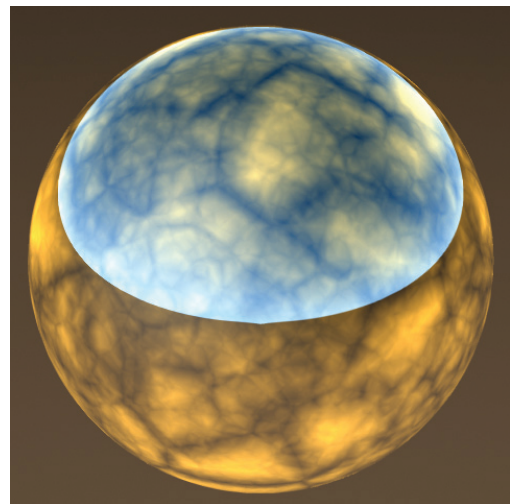
### Edit Panel

None

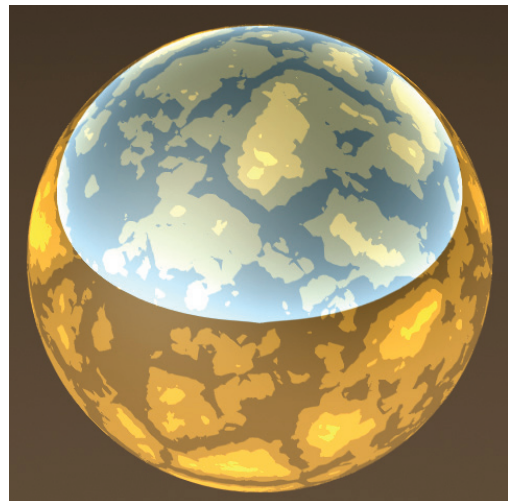
### Usage Example

As mentioned one of the possible uses for the Ceil node is to step otherwise smooth data from other sources; kind of like you see in posterization effects in some paint programs and compositing applications. In this example we will take the smooth value gradations created by the Crackle textures alpha channel and step them into only four values and apply that to the luminosity channel of our test sphere texture.

Here's what the surface would look like before we applied Ceil just connecting the Crackle Alpha output directly to the luminosity channel.

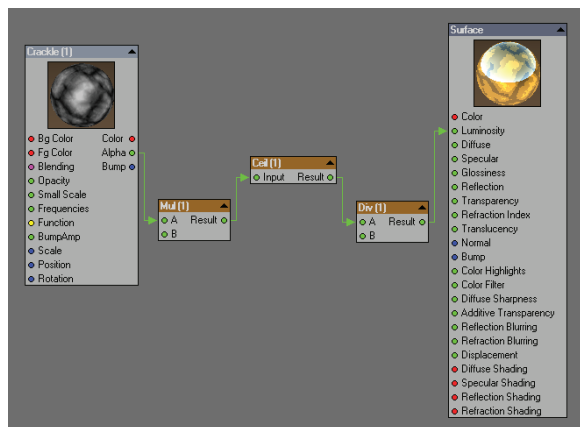


And here is the same surface with the Ceil node operating on the alpha output of the Crackle texture node.



As you can see the gradient smoothness that Crackle produces has been flattened into just four levels or “steps” as we referred to them here but the general Crackle pattern hasn't changed.

This was extremely easy to setup in the Node Editor and would have been nearly impossible to achieve without its use. Let's take a look at how the effect was accomplished.



Alpha information in the texturing world is represented as floating point values and unmodified by opacity levels and so forth, ranges from 0.0 to 1.0 which when passed through the Ceil node would only leave values of zero at zero and all other values between 0.0 and 1.0 would become 1.0. So in practical application for this texture, all resulting values would be 1.0 or 100% luminosity. This is not what we want. We need to stretch out the value range that Ceil receives so that it has bigger numbers than 1.0. Since we're after four levels (or only four values) we multiply the output from Crackles alpha by 4 giving use a value spread from 0 to 4. This is achieved by adding a Math > Multiply node, entering a value of 4 for the B value, and connecting the alpha from the Crackle texture to the Multiply node's A input.

This will work and will get Ceil to give us our four values (1, 2, 3 and 4) for the respective ranges. But a value of 4 plugged directly into the Luminosity channel means 400% luminosity will be applied to the surface in those areas. Values of 3 will be 300% and so on. This isn't what we're after either. We need to scale the data back down to a value range of 0..1.

This is easily done by adding a Division node and dividing the Ceil Result by the same number we multiplied it by - in this case 4. The results of the division are 0.25, 0.50, 0.75 and 1.00. giving us surface areas that are 25%, 50%, 75% and 100% luminous respectively.

While this certainly isn't the only reason you might make use of the Ceil node it gives a good illustration of the function of the Ceil node and demonstrates one possible use for it.



NOTE: By changing the values in B for both the Multiply and Divide nodes to some number of your choosing you can control the number of levels in the posterization effect. Additionally you can use other nodes to control both B values and texture the posterization or posterize by incidence angle, or etc.

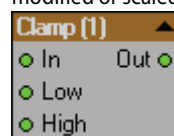
## Clamp

### Purpose

To clamp or hold incoming scalar values within a range specified by high and low levels and output the clipped range for use in other nodes.

### Description

This node will clamp incoming values to a user specified cap or limiter value. There is a cap for the high limit (or top of the range) and one for the low. Values that are greater than the specified high limit will be set to the high limit value. Values that are lower than the low limit will be set to the low limit. Incoming values that are between the specified high and low limits will not be modified or scaled.



### Inputs

In (type: scalar):

This input receives scalar values from other nodes. This is the incoming range that will be clamped to the high and low values specified.

Low (type: scalar):

The low value or bottom limit of the clamp can be specified by user input in the text entry field in the Node Edit panel or can be acquired from the output of another node by connecting another node's scalar output.

High (type: scalar):

The High value or top limit of the clamp can be specified by user input in the text entry field in the Node Edit panel or can be acquired from the output of another node by connecting another node's scalar output.



## Outputs

### Out (type: scalar):

The results of the clamp operation can be connected to other node for further processing or directly connected to the Surface Destination node.

### Edit Panel



### Usage Example

For this example let's start with a texture node that will allow us to easily understand what is taking place in the Clamp node. By using the Ripples texture node and setting the frequency value to 1 we should see a single set of concentric rings with very smooth gradients from ring to ring. We will use the Clamp tool here to clamp those smooth rings into higher contrast stripe-like bands.

If you can imagine the luminous values of these rings plotted on a 2D graph they would probably closely resemble the appearance of a sine wave. What we are about to do is flatten the peaks and valleys of that sine wave by clamping the high values lower and the low values higher.

In order to reproduce this example add a Clamp node and set the lower value to 0.4 and the high value to 0.6 by double clicking on the node and entering the values in the proper locations. Add a Ripples node and edit it in the same way entering 1 for the number of frequencies and setting the scale values X, Y, and Z, to something appropriate for the surface of the object you have loaded. Here in the X, Y, and Z. values should be approximately one tenth the greatest dimensional width of the object you chose to work with. It is recommended creating a default ball in modeler.

Here is what the test ball looks like with the ripples Alpha channel connected directly to the luminosity input of the destination surface node.



Notice here that without the clamp node affecting the values that the colors smoothly transition from yellow to black to yellow and so on.

Now let's see what happens when we connect the Alpha channel of the ripples node to the Clamp node Input connection and the Clamp nodes Result output to the Surface destination Luminosity channel.



As you can see we did indeed get the flattened contrasting stripes that we were expecting from this particular usage of the Clamp node.

Here is a screen grab of the actual node construction in the work area so you can see for yourself just exactly how easy it is to set up very powerful node structure combinations.



You will find that the Clamp node is an extremely useful tool for isolating patterns in textures or clipping out unwanted detail from the images, textures, and other data types manipulatable in the Node Editor. Additionally if you're making heavy use of the node editors mathematical functions the Clamp node can be used to insure that your results are always within a specific range.



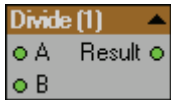
## Divide

### Purpose

The Divide Math function provides a divide-by function.

### Description

The Divide node allows one scalar to be divided by another.



### Inputs

**A** (Type: Scalar):

The A input provides the dividend.

**B** (Type: Scalar):

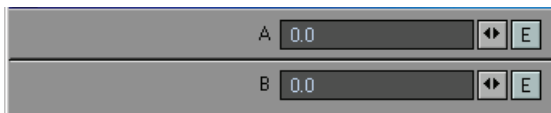
The B input provides the divisor.

### Outputs

**Result** (Type: Scalar):

The Result output is the A input divided by the B input.

#### Edit Panel



## Floor

### Purpose

The Floor Math function provides a standard floor function.

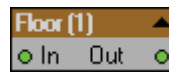
### Description

The Floor node finds the largest integer less than or equal to the In input. A few examples:

**In:**     -2.9 -2.0 -1.5  0.0 0.5  1.5 2.0 2.9

**Out:**    -3  -2  -2  0  0  1  2  2

The floor function can be considered as a round function much like Ceil, with the exception that it is derived from the lowest integer (the floor) rather than the highest integer (the ceiling).



### Inputs

**In** (Type: Scalar):

The Floor node will derive the floor of the In input.

### Outputs

**Out** (Type: Scalar):

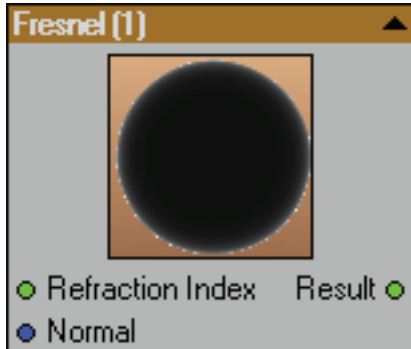
The results of the Floor operation can be connected to other nodes for further processing or directly connected to the Surface destination node.



## Fresnel

### Purpose

The Fresnel equation involves the amount of reflected light based on the incidence angle.



### Inputs

#### Refraction Index

Refraction is the bending of light as it passes through transparent materials or substances of different density.

Dense substances bend light more than materials that are not dense.

See elsewhere in this manual for a list of many real world transparent materials and their corresponding IOR values.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node

#### Normal (Type: Vector):

The Normal is the normal for smooth, bump-mapped, shaded polygons under the current spot of evaluation.

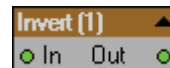
## Invert

### Purpose

The Invert Math function provides a standard inversion function.

### Description

The invert node simply inverts the input. This is done by subtracting the In input from 1.0, so 0.0 will become 1.0 and 1.0 will become 0.0. The inversion will occur for intermediate values also, so 0.2 inverted is 0.8, 0.7 inverted is 0.3 and so on.



### Inputs

#### In (Type: Scalar):

The input to be inverted.

### Outputs

#### Out (Type: Scalar):

The Out output is driven with the inversion in the input (i.e.  $Out = 1.0 - In$ ).

### Edit Panel

None.



## Logic

### Purpose

To provide conditional flow or result, within a node network.

### Description

The Logic node can be used for typical “If, then, else” logic flow within a network. There are two inputs that can be compared in various ways. If the given comparison evaluates true then the value entered or connected to the true input is passed to the Out output. If the comparison evaluates as false then the value entered or connected to the False input is passed to the Out output.

### Inputs

**A** (Type: scalar):

Allows an input to be connected or a values to be entered via the Edit panel, that will be compared with the value entered or connected to B. How the two values are compared depends on the Operation that is selected in the Edit panel of the Logic node.

**B** (Type: scalar):

The B input can be connected or a value to be entered via the Edit, panel that will be compared with the value entered or connected to A. How the two values are compared depends on the Operation that is selected in the Edit panel of the Logic node.

**True** (Type: scalar):

If the selected Operation results in a true condition then the values entered here or the input connected to the True input will be passed to the Out output of this node.

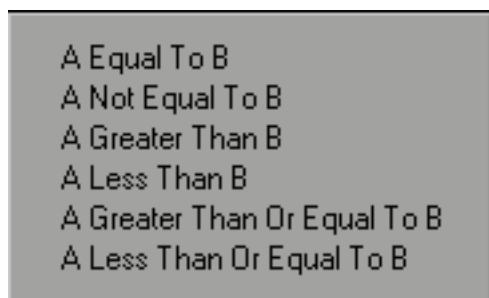
**False** (Type: scalar):

If the selected Operation results in a false condition then the values entered here or the input connected to the False input will be passed to the Out output of this node.

### Outputs

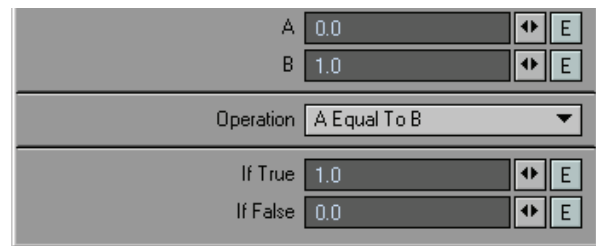
**Out** (Type: scalar):

The Out output from this node is either a pass-through of the True or False connections or the value entered into those variables via the Logic node's Edit panel. Which input True or False is passed through depends on the result of the selected Operation. The available Operations are shown here below:



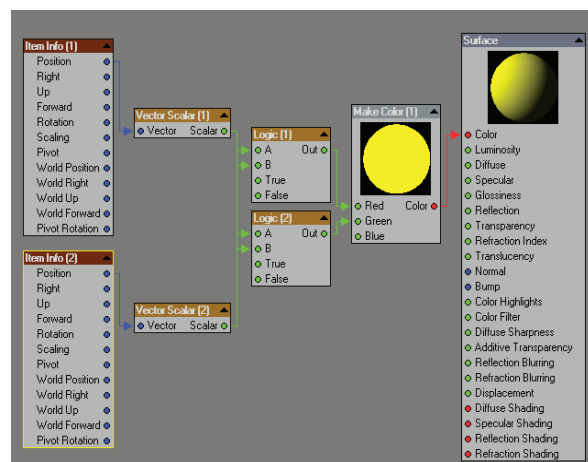
So for example if “A Equal To B” were selected the True value would be output if the vales for A and B were the same. If they were different then the value entered or connected to False would be passed to the Out output.

### Edit Panel



### Usage Example

Controlling a traffic light with two Logic nodes.



In this example the Y position of a Null parented to a traffic light object is used to control the color of the traffic light. Here two Item Info nodes are being used to get the position of a Null and the Traffic Light object. Vector Scalar nodes are being used to examine just the Y value of the two item' positions. The Y positions are each plugged into each of the two Logic nodes. One for the A input and one for the B input. The “Greater Than Or Equal To” Operator was selected in Logic (1) above and the “Less Than Or Equal To” Operator was selected in Logic (2). True was set to 1.0 and False was set to 0.0 for both of the Logic nodes and the Out of Logic (1) was connected to the Red input of the Make Color node while the Out of Logic (2) was connected to the Green input. This is really much simpler than it's description. The statement for programmers would be coded something like:

```
surface.color = (red*(a.pos > b.pos) + green*(a.pos < b.pos))
```

And basically means that if the Null Object is higher (Y) than the TrafficLight Object then the color is red. If the Null object is lower than the TrafficLight Object then the color is green and if the two objects are the same then both Red and Green are turned on making the color yellow.





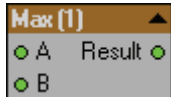
## Max

### Purpose

The Max Math function provides a standard maximum function.

### Description

The Max node simply takes the A and B inputs and outputs the largest. A and B may be connected to other nodes or controlled manually. This means that the node can be used to clamp the maximum value of a scalar.



### Inputs

A (Type: Scalar):

B (Type: Scalar):

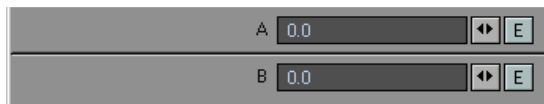
The node will derive the largest, or maximum, of the A and B inputs.

### Outputs

Out (Type: Scalar):

The Out output is driven with the largest of the A and B inputs.

### Edit Panel



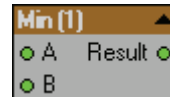
## Min

### Purpose

The Min Math function provides a standard minimum function.

### Description

The Min node simply takes the A and B inputs and outputs the smallest. A and B may be connected to other nodes or controlled manually. This means that the node can be used to clamp the minimum value of a scalar.



### Inputs

A (Type: Scalar):

B (Type: Scalar):

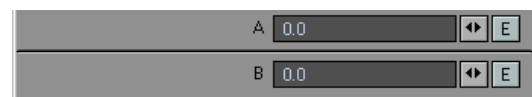
The node will derive the smallest, or minimum, of the A and B inputs.

### Outputs

Out (Type: Scalar):

The Out output is driven with the smallest of the A and B inputs.

### Edit Panel





## Mod

### Purpose

The Mod Math function provides a standard modulus function.

### Description

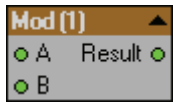
The Mod node derives the modulus of the A input in the form  $\text{Result} = A \bmod B$ . The modulus is effectively the remainder leftover when A is divided by B, as in 7 divided by 2 is 3 with a remainder 1. So a few examples would be:

Input A: 0.0 1.0 2.0 3.0 4.0 5.0 6.0

Input B: 2.0 2.0 2.0 2.0 2.0 2.0 2.0

Result: 0.0 1.0 0.0 1.0 0.0 1.0 0.0

This function is useful because it can be used to repeat a regular function over a shorter period. Take, for example, a linear gradient that varies from 0.0 at the top of an object to 6.0 at the bottom. By using this gradient as the A input and making B 2.0, you can see from the above table that the gradient would ramp from 0.0 to 1.0 three times.



### Inputs

A (Type: Scalar):

The A input has its modulus taken.

B (Type: Scalar):

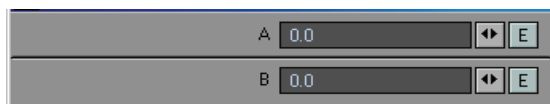
The B input is the modulus for the function.

### Outputs

Out (Type: Scalar):

The Out output is driven with  $A \bmod B$ .

### Edit Panel



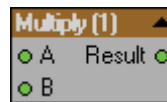
## Multiply

### Purpose

The Multiply Math function provides a multiply function.

### Description

The Multiply node allows one scalar to be multiplied by another.



### Inputs

A (Type: Scalar):

The A input provides the multiplicand.

B (Type: Scalar):

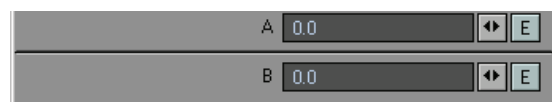
The B input provides the multiplier.

### Outputs

Result (Type: Scalar):

The Result output is the A input multiplied by the B input.

### Edit Panel





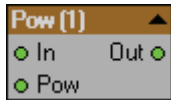
## Pow

### Purpose

The Pow Math function provides a “raise to the power of” function.

### Description

The Pow node allows one scalar to be raised to the power of another (A Pow B or AB). For example, 23 is  $2 \times 2 \times 2 = 8$ . Interestingly, you can also raise a scalar to the power of 0.5 if you wish to calculate the square root.



### Inputs

A (Type: Scalar):

B (Type: Scalar):

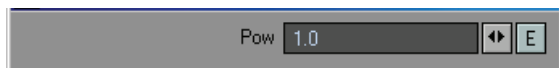
The A input is raised to the power of the B input.

### Outputs

Result (Type: Scalar):

The Result output is the A input raised to the power of the B input.

### Edit Panel



## Sign

### Purpose

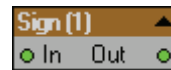
The Sign Math function provides a sign function.

### Description

The Sign node simply changes the sign of the input so that positive scalars become negative and negative scalars positive.

In:        -2.0        -1.0        -0.5        0.0        0.5        1.0  
2.0

Out:       2.0        1.0        0.5        0.0        -0.5       -1.0  
-2.0



### Inputs

In (Type: Scalar):

The sign node swaps the sign of this input.

### Outputs

Result (Type: Scalar):

The Result output is the magnitude of the input but with its sign swapped.

### Edit Panel

None



## Smooth Step

### Purpose

The Smooth Step Math function provides a standard smooth step function that can be applied to any scalar.

### Description

A smooth step is a rounded, ramped step function. Two limits are set by the Begin and End inputs. If the In input is less than the Begin input, then the output will be 0.0. Conversely, if the In input is greater than the End input, then the output will be 1.0. If the input is between Begin and End, then the output will vary between 0.0 and 1.0 proportionally as a smoothed ramp. Effectively the Smooth Step clamps between the begin and end values and normalizes the output between 0.0 and 1.0 with a smoothed transition.



### Inputs

**In** (Type: Scalar):

In is the input to the smooth step function.

**Begin** (Type: Scalar):

The Begin input specifies the start of the smooth step ramp. If In is below this value 0.0 is output. If In between Begin and End input the output will vary between 0.0 and 1.0, proportionally.

**End** (Type: Scalar):

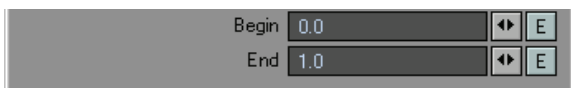
The End input specifies the end of the smooth step ramp. If In is above this value 1.0 is output. If In between Begin and End input the output will vary between 0.0 and 1.0, proportionally.

### Outputs

**Out** (Type: Scalar):

The Out output will vary between 0.0 and 1.0 as the input varies between the values of Begin and End.

### Edit Panel



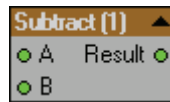
## Subtract

### Purpose

The Subtract Math function provides a subtract function.

### Description

The Subtract node allows one scalar to be subtracted from another.



### Inputs

**A** (Type: Scalar):

The A input provides the minuend.

**B** (Type: Scalar):

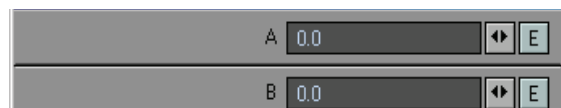
The B input provides the subtrahend.

### Outputs

**Result** (Type: Scalar):

The Result output is the B input subtracted from the A input.

### Edit Panel





## Trigonometry

### ArcCos

#### Purpose

The ArcCos Trigonometric function provides a standard inverse cosine function that can be applied to any scalar.

#### Description

The ArcCos node takes the inverse cosine of the input scalar.

The input can be in the range -1.0 to 1.0 for a real result. The result will be in the range  $\pi$  to 0.

If the input is not in range, the result will be an invalid floating point value.

Example:  $\cos(x) = y$  implies  $\arccos(y) = x$



#### Inputs

In (Type: Scalar):

The inverse cosine (ArcCos) is taken of this input.

#### Outputs

Out (Type: Scalar):

This output is the inverse cosine (ArcCos) of the input.

### ArcSin

#### Purpose

The ArcSin Trigonometric function provides a standard inverse sine function that can be applied to any scalar.

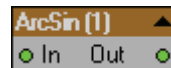
#### Description

The ArcSin node takes the inverse sine of the input scalar.

The input can be in the range -1.0 to 1.0 for a real result. The result will be in the range  $-\pi/2$  to  $\pi/2$ .

If the input is not in range, the result will be an invalid floating point value.

Example:  $\sin(x) = y$  implies  $\arcsin(y) = x$



#### Inputs

In (Type: Scalar):

The inverse sine (ArcSin) is taken of this input.

#### Outputs

Out (Type: Scalar):

This output is the inverse sine (ArcSin) of the input.



## ArcTan

---

### Purpose

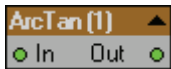
The ArcTan Trigonometric function provides a standard inverse tangent function that can be applied to any scalar.

### Description

The ArcTan node takes the inverse tangent of the input scalar.

The input can be any scalar value. The result will be in the range  $-\pi/2$  to  $\pi/2$ .

Example:  $\tan(x) = y$  implies  $\arctan(y) = x$



### Inputs

In (Type: Scalar):

The inverse tangent (ArcTan) is taken of this input.

### Outputs

Out (Type: Scalar):

This output is the inverse tangent (ArcTan) of the input.

### Edit Panel

None.

## Cos

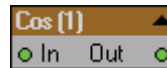
---

### Purpose

The Cos Trigonometric function provides a standard cosine function that can be applied to any scalar.

### Description

The Cos node takes the cosine of the input scalar.



### Inputs

In (Type: Scalar):

The cosine is taken of this input.

### Outputs

Out (Type: Scalar):

This output is the cosine of the input.

### Edit Panel

None.





## Sin

---

### Purpose

The Sin Trigonometric function provides a standard sine function that can be applied to any scalar.

### Description

The Sin node takes the sine of the input scalar.



### Inputs

In (Type: Scalar):

The sine is taken of this input.

### Outputs

Out (Type: Scalar):

This output is the sine of the input.

### Edit Panel

None.

## Tan

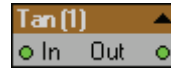
---

### Purpose

The Tan Trigonometric function provides a standard tangent function that can be applied to any scalar.

### Description

The Tan node takes the tangent of the input scalar.



### Inputs

In (Type: Scalar):

The tangent is taken of this input.

### Outputs

Out (Type: Scalar):

This output is the tangent of the input.

### Edit Panel

None.



## Vector

### Add

#### Purpose

The Add Vector function provides a standard add function that can add together two vectors.

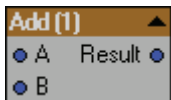
### Description

The Add node simply adds two vectors. Vectors are assumed to contain three elements X, Y and Z. This node adds the vectors so that:

$$\text{ResultX} = \text{AX} + \text{BX}$$

$$\text{ResultY} = \text{AY} + \text{BY}$$

$$\text{ResultZ} = \text{AZ} + \text{BZ}$$



### Inputs

**A** (Type: Vector):

**B** (Type: Vector):

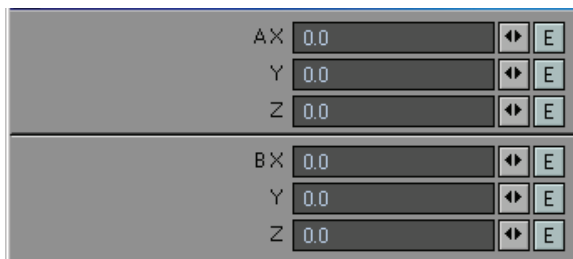
These are the two input vectors that will be added.

### Outputs

**Out** (Type: Vector):

This output is the result of vector A plus vector B.

### Edit Panel



## Add4

### Purpose

The Add4 Vector function provides an add function that can add together four vectors.

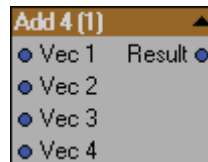
### Description

The Add4 node simply adds together four vectors. Vectors are assumed to contain three elements X, Y and Z. This node adds the four input vectors together so that:

$$\text{ResultX} = \text{Vec 1X} + \text{Vec 2X} + \text{Vec 3X} + \text{Vec 4X}$$

$$\text{ResultY} = \text{Vec 1Y} + \text{Vec 2Y} + \text{Vec 3Y} + \text{Vec 4Y}$$

$$\text{ResultZ} = \text{Vec 1Z} + \text{Vec 2Z} + \text{Vec 3Z} + \text{Vec 4Z}$$



### Inputs

**Vec 1** (Type: Vector);, **Vec 2** (Type: Vector);, **Vec 3** (Type: Vector);, **Vec 4** (Type: Vector)

These are the four input vectors to be added together.

### Outputs

**Out** (Type: Vector):

This output is the product of Vec 1, Vec 2, Vec 3, and Vec 4.



## Add Scaled

### Purpose

The Add Scaled Vector function provides an add-then-scale function that can add together two vectors and scale the result.

### Description

The Add Scale node adds two vectors and then scales the result. The scale works based on a percent, which means a scale of 100% is the same as multiplying by 1.0. A scale of 0% is the same as multiplying by 0.0, and a scale of -100% is the same as multiplying by 0.0. Vectors are assumed to contain three elements X, Y, and Z. This node adds the A and B vectors together and scales the result so that:

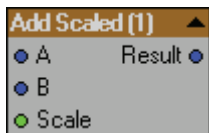
$$\text{ResultX} = (AX + BX) \times \text{Scale}$$

$$\text{ResultY} = (AY + BY) \times \text{Scale}$$

$$\text{ResultZ} = (AZ + BZ) \times \text{Scale}$$

Where:

$$\text{Scale} = ((\% \text{ Scale Input}) / 200.0) + 0.5$$



### Inputs

**A** (Type: Vector); **B** (Type: Vector):

These are the two input vectors that will be added to form the result, which is then scaled.

**Scale** (Type: Scalar):

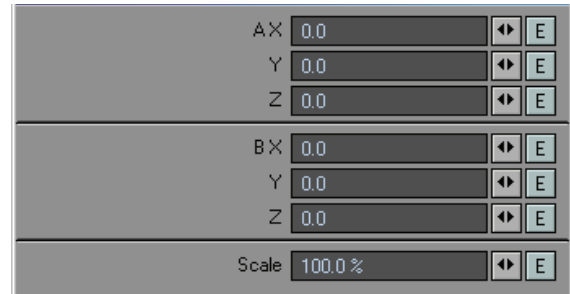
The scale input scales each vector element of the addition. As shown above, Scale is based on a percentage.

### Outputs

Result (Type: Vector):

This output is the scaled result from vector A plus vector B.

### Edit Panel





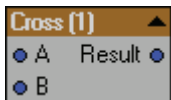
## Cross

### Purpose

The Cross Vector function provides a standard vector cross product function.

### Description

A vector cross product calculates the result as being the product vector orthogonal to both input vectors. Imagine two vectors X and Y on a plane where Y points up the plane and X across it. The cross product would be a vector that points directly out of the plane, which is otherwise referred to as the Z vector (this can also be considered the normal to the plane).



### Inputs

A (Type: Vector);, B (Type: Vector):

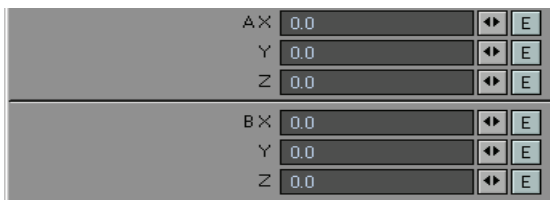
These are the two input vectors to be crossed.

### Outputs

Result (Type: Vector):

The Result output is the cross product for the two input vectors.

### Edit Panel



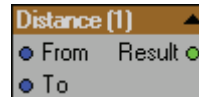
## Distance

### Purpose

The Distance Vector function derives the distance between two points.

### Description

The distance node derives the distance between the two input points To and From.



### Inputs

From (Type: Vector):

To (Type: Vector):

These two inputs define the two points from which the distance is determined.

### Outputs

Result (Type: Scalar):

The Result output is the distance between the two points To and From.

### Edit Panel

None



## Divide

### Purpose

The Divide Vector function provides a standard divide function that can divide one vector by another.

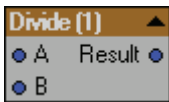
### Description

The Divide node simply divides one vector by another. Vectors are assumed to contain three elements X, Y and Z. This node divides the vectors so that:

$$\text{ResultX} = \text{AX} / \text{BX}$$

$$\text{ResultY} = \text{AY} / \text{BY}$$

$$\text{ResultZ} = \text{AZ} / \text{BZ}$$



### Inputs

**A** (Type: Vector):

The A input provides the dividend vector.

**B** (Type: Vector):

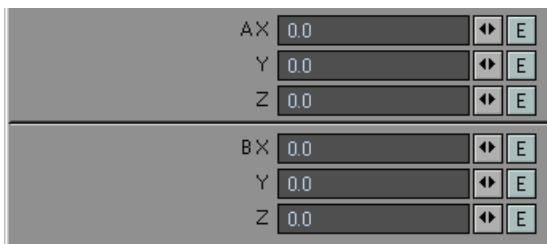
The B input provides the divisor vector.

### Outputs

**Out** (Type: Vector):

This output is the result of vector A divided by Vector B.

### Edit Panel



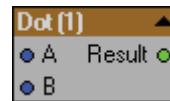
## Dot

### Purpose

The Purpose Vector function provides a standard vector dot product function.

### Description

A vector dot product calculates the cosine of the angle between two vectors. The dot product ranges from -1.0 to 1.0. Taking the inverse cosine of this output yields the angle between the two input vectors.



### Inputs

**A** (Type: Vector); **B** (Type: Vector):

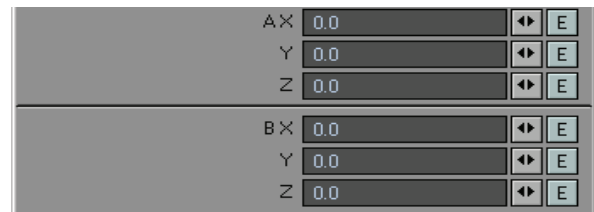
The dot product is found as the cosine of the angle between these two input vectors.

### Outputs

**Result** (Type: Vector):

The Result output is the dot product for the two input vectors.

### Edit Panel





## Length

### Purpose

The Length Vector function provides a function that finds the length of a vector.

### Description

The Length node determines the length of any vector.



### Inputs

**Vector** (Type: Vector):

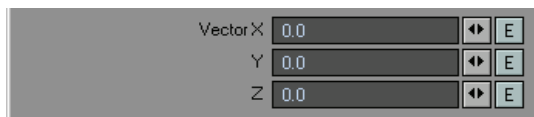
The Vector input receives the vector value for its length to be calculated.

### Outputs

**Result** (Type: Vector):

The Result output is the length of the input vector.

### Edit Panel



## Multiply

### Purpose

The Multiply Vector function provides a standard multiply function that can multiply two vectors together.

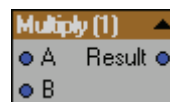
### Description

The Multiply node simply multiplies one vector by another. Vectors are assumed to contain three elements X, Y and Z. This node multiplies the vectors so that:

$$\text{ResultX} = \text{AX} \times \text{BX}$$

$$\text{ResultY} = \text{AY} \times \text{BY}$$

$$\text{ResultZ} = \text{AZ} \times \text{BZ}$$



### Inputs

**A** (Type: Vector):

The A input provides the multiplicand vector.

**B** (Type: Vector):

The B input provides the multiplier vector.

### Outputs

**Out** (Type: Vector):

This output is the result of vector A multiplied by Vector B.

### Edit Panel







## Normalize

### Purpose

The Normalize Vector function provides a standard vector normalization function.

### Description

Vector normalization involves taking a vector and finding its length. All three elements of the vector are then divided by this length. This has the effect of creating a normalized vector, that is, a vector with the same direction as the input vector but with a unit length.



### Inputs

**Vector** (Type: Vector):

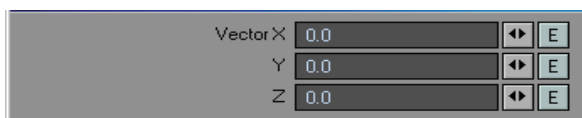
The Vector input is the vector to be normalized.

### Outputs

**Result** (Type: Vector):

This output is the normalized version of the input vector.

### Edit Panel



## Scale

### Purpose

The Scale Vector function provides a function that allows a vector to be scaled.

### Description

The scale function simply scales the size of the vector. The scale works based on a percent, which means a scale of 100% is the same as multiplying by 1.0. A scale of 0% is the same as multiplying by 0.5, and a scale of -100% is the same as multiplying by 0.0. Vectors are assumed to contain three elements X, Y and Z. This node scales the input vector so that:

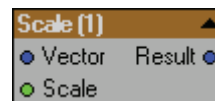
$$\text{ResultX} = \text{VectorX} \times \text{Scale}$$

$$\text{ResultY} = \text{VectorY} \times \text{Scale}$$

$$\text{ResultZ} = \text{VectorZ} \times \text{Scale}$$

Where:

$$\text{Scale} = ((\% \text{ Scale Input}) / 200.0) + 0.5$$



### Inputs

**Vector** (Type: Vector):

This is the input vector to be scaled.

**Scale** (Type: Scalar):

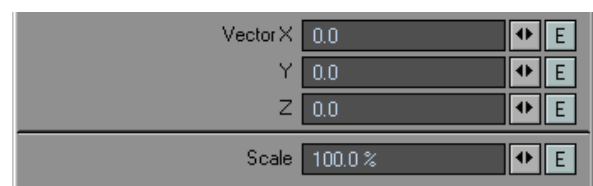
The scale input scales each element of the input vector as described above.

### Outputs

**Result** (Type: Vector):

This output is the scaled input Vector.

### Edit Panel





## Subtract

### Purpose

The Subtract Vector function provides a standard subtract function that can subtract one vector from another.

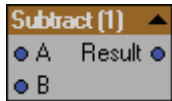
### Description

The Subtract node simply subtracts one vector from another. Vectors are assumed to contain three elements X, Y and Z. This node subtracts the vectors so that:

$$\text{ResultX} = \text{AX} - \text{BX}$$

$$\text{ResultY} = \text{AY} - \text{BY}$$

$$\text{ResultZ} = \text{AZ} - \text{BZ}$$



### Inputs

**A** (Type: Vector):

The A input provides the minuend vector.

**B** (Type: Vector):

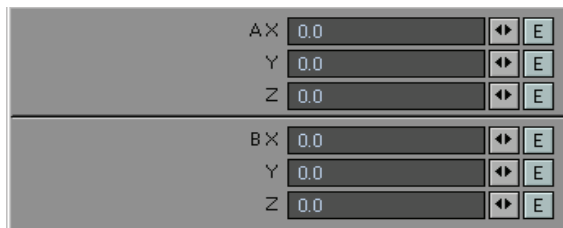
The B input provides the subtrahend vector.

### Outputs

**Out** (Type: Vector):

This output is the result of vector A minus vector B.

### Edit Panel



## Subtract Scaled

### Purpose

The Subtract Scaled vector function provides a subtract-then-scale function that can subtract one vector from another and scale the result.

### Description

The Subtract Scale node subtracts two vectors, and then scales the result. The scale works based on a percent, which means a scale of 100% is the same as multiplying by 1.0. A scale of 0% is the same as multiplying by 0.5, and a scale of -100% is the same as multiplying by 0.0. Vectors are assumed to contain three elements X, Y and Z. This node subtracts the A and B vectors and scales the result so that:

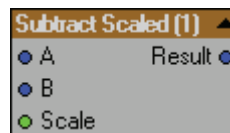
$$\text{ResultX} = (\text{AX} - \text{BX}) \times \text{Scale}$$

$$\text{ResultY} = (\text{AY} - \text{BY}) \times \text{Scale}$$

$$\text{ResultZ} = (\text{AZ} - \text{BZ}) \times \text{Scale}$$

Where:

$$\text{Scale} = ((\% \text{ Scale Input}) / 200.0) + 0.5$$



### Inputs

**A** (Type: Vector):

The A input provides the minuend vector.

**B** (Type: Vector):

The B input provides the subtrahend vector.

**Scale** (Type: Scalar):

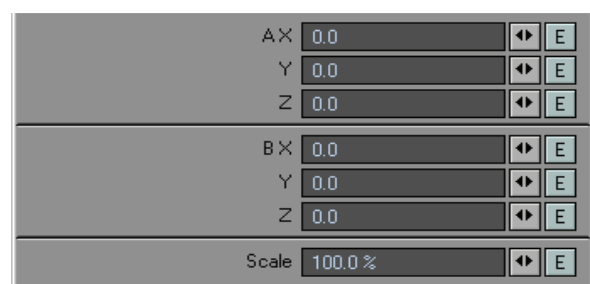
The scale input scales each element of the result of the subtraction. As show above, Scale is based on a percentage.

### Outputs

**Result** (Type: Vector):

This output is the scaled result from vector A minus vector B.

### Edit Panel





## Transform

### Purpose

The Transform Vector function provides a method of translating a vector between world and object coordinate systems.

### Description

The Transform node allows a vector in object space to be transformed into the same vector in world space. This may also be applied in reverse, so that a vector in world space can be transformed into object space.



### Inputs

**Vector** (Type: Vector):

The Vector input provides vector to be transformed.

**Type** (Section: Object to World or World to Object):

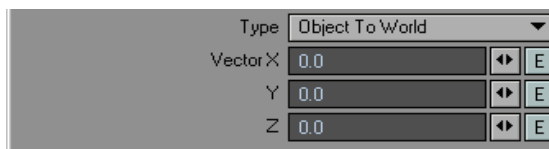
This input allows the transformation (object to world or world to object) to be selected.

### Outputs

**Out** (Type: Vector):

This output is the transformed vector.

### Edit Panel



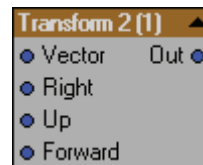
## Transform2

### Purpose

The Transform2 Vector function provides a method of translating a vector using a user defined matrix.

### Description

The Transform2 node allows a vector to be translated using a 3x3 transformation matrix as defined by the user. Here, the Right input defines the top row of the transformation matrix, the Up vector the middle row, and the Forward vector the bottom row. A switch is provided so that the inverse transform can be performed using the same transformation matrix.



### Inputs

**Vector** (Type: Vector):

The Vector input provides the vector to be transformed.

**Right, Up Forward** (Type: Vector):

Three input vectors that define the three row of the transformation matrix, respectively.

**Inverse Transform** (Select On or Off)

An input switch that allows the inverse transform to be performed.

### Outputs

**Out** (Type: Vector):

This output is the input vector transformed by the user defined transformation matrix.

### Edit Panel





## RayTrace

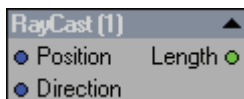
### RayCast

#### Purpose

The RayCast RayTrace function allows rays to be cast into the scene from any point in any direction.

#### Description

The RayCast node allows rays to be cast into the scene. The rays are "fired" from the Position input vector and will travel in the direction of the Direction input vector. This is the fastest way of tracing rays through the scene, as the ray casting only returns the length the ray traveled before it hits another surface (on any object including the objects whose hit point is currently being evaluated). Both the Position and Direction inputs must be specified in world coordinates (the Transform and Transform2 nodes can be used as required). If the ray fired doesn't hit another object, then the Length is returned as being -1.0.



#### Inputs

**Position** (Type: Vector):

The Position input defines the point, specified in world coordinates, from which the ray is fired.

**Direction** (Type: Vector):

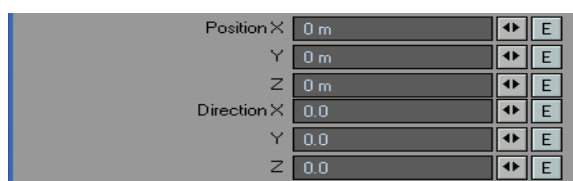
The Direction input vector determines the direction in which the ray is fired. Again, this must be specified in world coordinates.

#### Outputs

**Length** (Type: Scalar):

The Length output is the distance the ray traveled before intersecting another surface. The ray makes no distinction on the object whose surface is hit. The length will be returned as -1.0 if no surface was hit.

#### Edit Panel



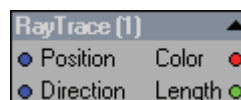
### RayTrace

#### Purpose

The RayTrace function allows rays to be traced through the scene from any point in any direction.

#### Description

The RayTrace node allows rays to be traced through the scene. The rays are "fired" from the Position input vector and will travel in the direction of the Direction input vector. As with the RayCast nodes, the ray trace function returns the length the ray traveled before it hit another surface. However, with RayTrace, the surface which the ray hits is also evaluated, and the result is returned as the Color output. This means that it may be used to trace reflection and refraction. Both the Position and Direction inputs must be specified in world coordinates (the Transform and Transform2 nodes can be used as required). If the ray fired doesn't hit another object, then the length is returned as being -1.0 and the color black.



#### Inputs

**Position** (Type: Vector):

The Position input defines the point, specified in world coordinates, from which the ray is fired.

**Direction** (Type: Vector):

The Direction input vector determines the direction in which the ray is fired. Again, this must be specified in world coordinates.

#### Outputs

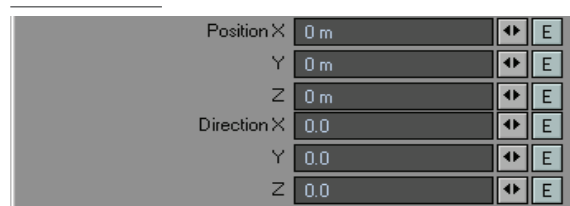
**Color** (Type: Color):

The Color output is driven with the color evaluated for the point on the surface hit by the traced ray. If no surface was hit this will be output as black.

**Length** (Type: Scalar):

This Length output is the distance the ray traveled before intersecting another surface. The ray makes no distinction on the object whose surface is hit. The length will be returned as -1.0 if no surface was hit.

#### Edit Panel





## Shaders (Diffuse)

Diffuse means to pour out and cause to spread freely. To spread about or scatter; disseminate. In the case of diffuse shaders within LightWave 3D, it is the reflection of light that is being scattered or spread out.

Various shading models are offered and each achieves this dissemination of light in a slightly different way resulting in a different overall surface "look" especially when animated. Some use trigonometric functions to scatter the rays while others may incorporate such technology as micro-geometry - a virtual layer of infinitesimally small geometric shapes used to calculate and vary the direction of reflected light rays.

Some of the diffuse shading models offered here are classic staples in the Computer Graphic Industry (CGI) while others are more unique. Several were designed specifically for LightWave 3D by Antti Järvelä, a renowned Finnish scientist and are exclusive to NewTek's LightWave 3D application.

The shading models offered in this section can additionally be mixed and modified in order to create a huge variety of different surface shading. Each significant shading model can shade to an individual set of normals thereby allowing true layered shading per surface.

For example one normal map can be connected to the Lambertian diffuse shader, an entirely different normal map connected to the Phong specular shader and then an additional third normal map for the surface globally by connecting it to the Surface destination nodes Normal input. There's no implied limit to the number of shaded Normal layers that can be added to a surface in this way. This is extremely useful for creating the illusion of depth within a surface be it car paint, very deep lacquer finishes, or more elaborate uses such as a freshly waxed surfboard, resin cup coasters, transparent signal cable jackets as seen below, and other photorealistic and non-photorealistic surfaces.



Multiple Normal mapped shading layers can be applied for each attribute; Diffuse, Reflection, Specular, Subsurface Scattering, and Transparent, besides the usual per Surface global normal mapping.

## Lambert

### Description

Lambertian diffuse shader. This node's output is intended to be connected to the Diffuse Shading input of the Surface destination node.

In LightWave3D's shading model prior to version 9.0 as well as in this node, Lambertian Reflection is used as a model for diffuse reflection. Lambertian Reflection as used here, is calculated by taking the dot product of the surface's normalized normal vector and a normalized vector pointing from the surface to the light source. This number is then multiplied by the color of the surface and the intensity of the light hitting the surface.

### Inputs

#### Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color used for the diffuse channel.

#### Diffuse (type: scalar):

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Specifies the amount of Diffuse reflectivity that will be applied.

#### Function (type: function):

Can receive inputs from other function nodes in the network only.

Connecting a function node to this input will transform the texture shading value of this shading model based on the graph defined by the function thereby extending the range of this nodes producible shading properties.

See the manual section on Function nodes for a description of how connecting functions can transform the values of the nodes they are connected to.

#### Normal (type: vector):

Normal maps connected to this input will replace the normals that this shader applies shading to in an individually exclusive manner from other shading models. This means that more than one set of normals can be utilized by any shaders exclusively or shared between any of the various shaders and/or the global normals for an individual surface.

The per-surface global normals are defined by either the interpolated geometry itself or by whatever is connected to the Normal input on the Surface destination node.

By using a different set of normals either from different normal maps or from any of various nodes that may offer normal outputs themselves, it becomes possible to create truly layered surface shading and texturing.

Can receive input from other vector type outputs that define surface normal information. Usually these outputs contain the word "Normal" in the output label text.



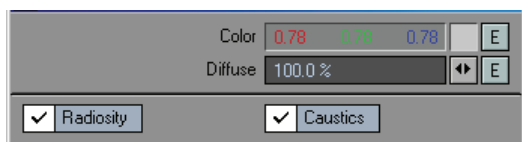
## Outputs

### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

Typically this output is connected to the Diffuse Shading input of the Surface destination node however, it may also be used within a network when using this shading model for other purposes or when mixing several diffuse shading models together.

### Edit Panel



Here the Radiosity and Caustics check boxes are not offered as node connections in the Workspace area and are features exclusive to the Edit Panel of the node.

When checked this shading model will evaluate shading information for the surface channel it is being applied including any Radiosity or Caustics computations as defined by those system attributes applied to the scene.

Unchecked, radiosity and/or caustics calculations will not be evaluated by this shading model as it is applied to the object surface being edited.

This can be used to omit both entire surfaces or per channel surface attributes from radiosity or caustics calculations thus offering more diversity in the rendered output as well as reducing the time needed to calculate these surfaces in scenes where either radiosity or caustics are used.

## Minnaert

### Description

Developed from the mathematical models of Marcel Gilles Jozef Minnaert (February 12, 1893 ~ October 26, 1970), a Belgian astronomer, for describing lunar or non-atmospherical terrain surfaces.

The air in our atmosphere affects what things look like. This is a shading model designed to render surfaces as they would appear without the atmosphere.

As a byproduct it is also commonly used for velvet like surfaces. The darkening lobes in this model are very well suited for rendering fabrics and soft materials such as velvet when viewed from intermediate distances.

### Inputs

#### Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color used for the diffuse channel.

#### Diffuse (type: scalar):

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Specifies the amount of Diffuse reflectivity that will be applied.

#### Darkening (type: scalar):

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Shadows the micro-geometry by increasing it's virtual height. High values can produce inverted shading.

#### Function (type: function):

Can receive inputs from other function nodes in the network only.

Connecting a function node to this input will transform the texture shading value of this shading model based on the graph defined by the function thereby extending the range of this nodes producible shading properties.

See the manual section on Function nodes for a description of how connecting functions can transform the values of the nodes they are connected to.





## Normal (type: vector):

Normal maps connected to this input will replace the normals that this shader applies shading to in an individually exclusive manner from other shading models. This means that more than one set of normals can be utilized by any shaders exclusively or shared between any of the various shaders and/or the global normals for an individual surface.

The per-surface global normals are defined by either the interpolated geometry itself or by whatever is connected to the Normal input on the Surface destination node.

By using a different set of normals either from different normal maps or from any of various nodes that may offer normal outputs themselves, it becomes possible to create truly layered surface shading and texturing.

Can receive input from other vector type outputs that define surface normal information. Usually these outputs contain the word "Normal" in the output label text.

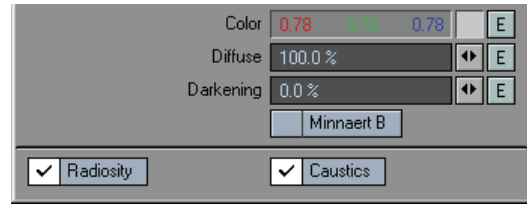
## Outputs

### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

Typically this output is connected to the Diffuse Shading input of the Surface destination node however, it may also be used within a network when using this shading model for other purposes or when mixing several diffuse shading models together.

## Edit Panel



The Radiosity and Caustics check boxes are not offered as node connections in the Workspace area and are features exclusive to the Edit Panel of the node.

When checked this shading model will evaluate shading information for the surface channel it is being applied including any Radiosity or Caustics computations as defined by those system attributes applied to the scene.

Unchecked, radiosity and/or caustics calculations will not be evaluated by this shading model as it is applied to the object surface being edited.

This can be used to omit both entire surfaces or per channel surface attributes from radiosity or caustics calculations thus offering more diversity in the rendered output as well as reducing the time needed to calculate these surfaces in scenes where either radiosity or caustics are used.

The Minnaert B check box is not offered as node connections in the Workspace area and is a feature exclusively accessed in the Edit Panel of this node.

Minnaert B offers alternative shading to the standard model and applies Darkening in a different way from the standard Minnaert shading model. Minnaert B attempts to equalize the effects of the Darkening component.



## Occlusion

### Description

Occlusion or “ambient occlusion” is a shading method used to help add realism by taking into account attenuation of light caused by obstructive geometry.

Local shading methods like Lambertian shading or the OrenNayar model do not take into account such global information; by contrast ambient occlusion is a global method even though it is a very crude approximation to full global illumination, considering only a visibility function and cosine application.

The appearance achieved by ambient occlusion is similar to the way an object appears on an overcast day.

This shader requires that Ray Trace Shadows be turned on in the Render Globals Render tab.

It was intended to be used as input to a wide variety of connections within any given network. For example as direct input to any number of channels (connections) on the Surface destination node, as input to the Opacity connection of any of the 2D or 3D Texture nodes or even as input to another Shaders' properties. Its use as intended, is limited only by your imagination with the few simple rules that apply to dissimilar connection types.

### Inputs

#### Samples (type: integer):

Can receive numerical inputs from other nodes in the network. Additionally, controls offered in the Edit Panel for this node may be used to specify this value.

Sampling is the number of directions the shading model examines the surface from in order to calculate the shading value of each spot on the surface under evaluation.

These readings or “Samples” are taken from positions on a hemispherical lattice and it is the number of longitudinal and latitudinal sections of this hemisphere that you are defining when you select one of the Sample preset values from the Samples pull down menu. It is for this reason that there are two numbers in each of the possible Sample selections.

It's quite easy to understand if you imagine one half of an unsmoothed (faceted) default ball object in modeler. The first number defines the number of longitudinal strips and the second number defines the number of latitudinal sections where each facet (or “face”) formed by their intersection defines one of the directions from which the spot is evaluated. You could visualize a sample size of 1x4 for example, as a bottomless pyramid centered over the evaluation spot protruding straight out from the surface normal - 1 segment high and containing 4 sides.

Higher values will sample the shading from more directions and produce smoother more accurate shading.

Incoming values higher than 15 will be set to 15. Incoming values lower than 0 will be set to 0.

#### Max (scalar):

Requires that the Mode be set to Ranged in the Edit Panel for this node. When Ranged Mode is selected evaluation sample lengths are limited to the specified Max value.

Occlusion shading works by firing a set of rays from the surface spot being evaluated. If a ray hits any other surface in the scene that ray is evaluated as occluded. In Ranged mode the ray will automatically be set to a non-occluded status after it has traveled the length specified as the Max limit.

### Outputs

#### Out (type: scalar):

The Occlusion shader can be used for a wide variety of purposes including the application of textures and/or to enhance Global Illumination. The Out output can be connected to any input where the properties of occlusion may be of benefit.

One very common technique is to set diffusion very low or at 0% and connect the output of the occlusion shader to the Luminosity input of the destination Surface node and then bake the surfaces for later post processing. See the information elsewhere in this manual that describes Surface Baking as well as the new Surface Baking Camera as a means to achieve this technique.

Outputs scalar values ranging from 0.0 to 1.0.

### Edit Panel



Mode is a property exclusive to the Edit Panel for this node and is not available as a input or output connection to or from the node as it appears in the workspace area.

When set to Infinite the Max value will be disabled in the Edit Panel for this node and any connections made to the Max input on the node in the workspace area will be ignored.



## Occlusion II

### Description

Occlusion or “ambient occlusion” is a shading method used to help add realism by taking into account attenuation of light caused by obstructive geometry.

Local shading methods like Lambertian shading or the OrenNayar model do not take into account such global information; by contrast ambient occlusion is a global method even though it is a very crude approximation to full global illumination, considering only a visibility function and cosine application.

The appearance achieved by ambient occlusion is similar to the way an object appears on an overcast day.

This shader requires that Ray Trace Shadows be turned on in the Render Globals Render tab.

It was intended to be used as input to a wide variety of connections within any given network. For example as direct input to any number of channels (connections) on the Surface destination node, as input to the Opacity connection of any of the 2D or 3D Texture nodes or even as input to another Shaders’ properties. It’s use as intended, is limited only by your imagination with the few simple rules that apply to dissimilar connection types.

### Edit Panel

### Inputs

### Samples

Can receive numerical inputs from other nodes in the network. Additionally, controls offered in the Edit Panel for this node may be used to specify this value.

Sampling is the number of directions the shading model examines the surface from in order to calculate the shading value of each spot on the surface under evaluation.

These readings or “Samples” are taken from positions on a hemispherical lattice and it is the number of longitudinal and latitudinal sections of this hemisphere that you are defining when you select one of the Sample preset values from the Samples pull down menu. It is for this reason that there are two numbers in each of the possible Sample selections.

It’s quite easy to understand if you imagine one half of an unsmoothed (faceted) default ball object in modeler. The first number defines the number of longitudinal strips and the second number defines the number of latitudinal sections where each facet (or “face”) formed by their intersection defines one of the directions from which the spot is evaluated. You could visualize a sample size of 1x4 for example, as a bottomless pyramid centered over the evaluation spot protruding straight out from the surface normal - 1 segment high and containing 4 sides.

Higher values will sample the shading from more directions and produce smoother more accurate shading.

For a table of the possible sampling matrices and the numerical values associated with each, see the end of this document.

### Mode

Mode is a feature exclusive to this nodes Edit Panel . Please see the explanation for these items in the chapter that covers texturing with the Surface Editor.

### Max

Requires that the Mode be set to Ranged in the Edit Panel for this node. When Ranged Mode is selected evaluation sample lengths are limited to the specified Max value.

Occlusion shading works by firing a set of rays from the surface spot being evaluated. If a ray hits any other surface in the scene that ray is evaluated as occluded. In Ranged mode the ray will automatically be set to a non-occluded status after it has traveled the length specified as the Max limit.

### Spread

The overall cone angle of the sampling rays.

### Heading

The rotation angle of the map for the spherical mapping mode, and also for the light probe mode.

### Pitch

The pitch rotation angle of the map, but is only applied for the light probe mode.

### Outputs

#### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

Typically this output is connected to the Diffuse Shading input of the Surface destination node however, it may also be used within a network when using this shading model for other purposes or when mixing several diffuse shading models together.



## OrenNayar

### Description

OrenNayar is a diffuse shading model proposed by Michael Oren and Shree Nayar of Columbia University in 1994, specifically as an improvement to the Lambertian model and especially good for simulating rough surfaces like clay where the darker areas tend to reflect light causing the surface to appear rougher. It is also particularly well suited to simulate cloth surfaces.

The CookTorrance specular model uses a similar method of distributing light by the distribution of a microfacet description.

### Inputs

#### Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color used for the diffuse channel.

#### Diffuse (type: scalar):

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Specifies the amount of Diffuse reflectivity that will be applied.

#### Roughness (type: scalar):

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Specifies the amount of roughness that will be applied. Roughness is the result of microfacet evaluation in lighting conditions. Higher values produce rougher surface properties while lower values produce smoother looking surfaces. This kind of roughness is more easily seen on objects with many sharp edges or where high contrast bump or normal maps are applied.

#### Function (type: function):

Can receive inputs from other function nodes in the network only.

Connecting a function node to this input will transform the texture shading value of this shading model based on the graph defined by the function thereby extending the range of this nodes producible shading properties.

See the manual section on Function nodes for a description of how connecting functions can transform the values of the nodes they are connected to.

#### Normal (type: vector):

Normal maps connected to this input will replace the normals that this shader applies shading to in an individually exclusive manner from other shading models. This means that more than one set of normals can be utilized by any shaders exclusively or shared between any of the various shaders and/or the global normals for an individual surface.

The per-surface global normals are defined by either the interpolated geometry itself or by whatever is connected to the Normal input on the Surface destination node.

By using a different set of normals either from different normal maps or from any of various nodes that may offer normal outputs themselves, it becomes possible to create truly layered surface shading and texturing.

Can receive input from other vector type outputs that define surface normal information. Usually these outputs contain the word "Normal" in the output label text.

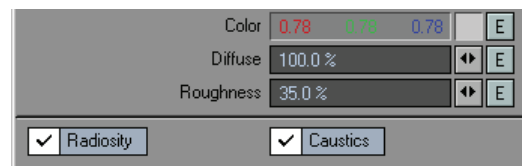
### Outputs

#### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

Typically this output is connected to the Diffuse Shading input of the Surface destination node however, it may also be used within a network when using this shading model for other purposes or when mixing several diffuse shading models together.

### Edit Panel



Here the Radiosity and Caustics check boxes are not offered as node connections in the Workspace area and are features exclusive to the Edit Panel of the node.

When checked this shading model will evaluate shading information for the surface channel it is being applied including any Radiosity or Caustics computations as defined by those system attributes applied to the scene.

Unchecked, radiosity and/or caustics calculations will not be evaluated by this shading model as it is applied to the object surface being edited.

This can be used to omit both entire surfaces or per channel surface attributes from radiosity or caustics calculations thus offering more diversity in the rendered output as well as reducing the time needed to calculate these surfaces in scenes where either radiosity or caustics are used.



## Theta

### Description

Theta is a superior LightWave3D exclusive, translucency shading model developed by Antti Järvelä, a Finnish Scientist. It has many of the same properties that people associate with Subsurface Scattering but without the intense rendering computations required by SSS shading models such as Omega and Kappa also included with Lightwave 3D.

Theta was especially designed for the translucent shading of thin walled objects. Such items as ping-pong balls, thin material lamp shades, sheets of paper, and so on.

Translucency is the material quality of allowing light to pass diffusely through semitransparent substances.

Since translucency is in consideration of light rays in specific, it is considered and classified as a diffuse shading model.

Translucent shaded surfaces will not reveal the surface colors and properties of other object surfaces that exist on the other side of the translucent object away from the observer. Lights however will show through translucent surfaces and are needed in order for translucency shaders to affect the surface at all.

Light and observer (usually the LightWave camera) positioning are critical to the way translucency affects the final rendered result.

This shader requires that Ray Trace Shadows be turned on in the Render Globals Render tab.

### Inputs

#### Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color used for the diffuse channel.

#### Amount (type: scalar):

Scales the shading values. This is the "amount" of the effect that this shader will have on the surface it is applied to.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

#### Spread (type: scalar):

Spread is the amount of angular spread of the scattering of light as it passes through the surface.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Consider a light located on the other side of a translucent object from a viewer. Coarser or finer translucent materials spread the light more or less and the light passes through the object on its way to the observer.

#### IOR (type: scalar):

Index Of Refraction or IOR, is a charted index of the different amounts of refraction associated with real world materials.

Refraction is the bending of light as it passes through transparent materials or substances of different density.

Dense substances bend light more than materials that are not dense.

See elsewhere in this manual for a list of many real world transparent materials and their corresponding IOR values.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

#### Function (type: function):

Can receive inputs from other function nodes in the network only.

Connecting a function node to this input will transform the texture shading value of this shading model based on the graph defined by the function thereby extending the range of this nodes producible shading properties.

See the manual section on Function nodes for a description of how connecting functions can transform the values of the nodes they are connected to.

#### Normal (type: vector):

Normal maps connected to this input will replace the normals that this shader applies shading to in an individually exclusive manner from other shading models. This means that more than one set of normals can be utilized by any shaders exclusively or shared between any of the various shaders and/or the global normals for an individual surface.

The per-surface global normals are defined by either the interpolated geometry itself or by whatever is connected to the Normal input on the Surface destination node.

By using a different set of normals either from different normal maps or from any of various nodes that may offer normal outputs themselves, it becomes possible to create truly layered surface shading and texturing.

Can receive input from other vector type outputs that define surface normal information. Usually these outputs contain the word "Normal" in the output label text.



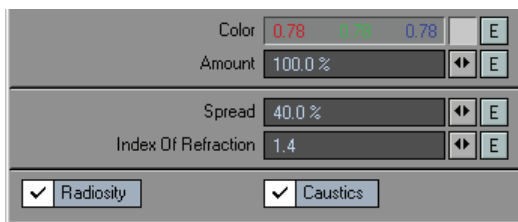
## Outputs

### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

Typically this output is connected to the Diffuse Shading input of the Surface destination node however, it may also be used within a network when using this shading model for other purposes or when mixing several diffuse shading models together.

#### Edit Panel



Here the Radiosity and Caustics check boxes are not offered as node connections in the Workspace area and are features exclusive to the Edit Panel of the node.

When checked this shading model will evaluate shading information for the surface channel it is being applied including any Radiosity or Caustics computations as defined by those system attributes applied to the scene.

Unchecked, radiosity and/or caustics calculations will not be evaluated by this shading model as it is applied to the object surface being edited.

This can be used to omit both entire surfaces or per channel surface attributes from radiosity or caustics calculations thus offering more diversity in the rendered output as well as reducing the time needed to calculate these surfaces in scenes where either radiosity or caustics are used.

## Translucency

### Description

Another LightWave3D exclusive translucency shading model developed by Antti Järvelä, a Finnish Scientist.

Translucency is the material quality of allowing light to pass diffusely through semitransparent substances.

Since translucency is in consideration of light rays in specific, it is considered and classified as a diffuse shading model.

Translucent shaded surfaces will not reveal the surface colors and properties of other object surfaces that exist on the other side of the translucent object away from the observer. Lights however will show through translucent surfaces and are needed in order for translucency shaders to affect the surface at all.

Light and observer (usually the LightWave camera) positioning are critical to the way translucency affects the final rendered result.

### Inputs

#### Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color used for the diffuse channel.

#### Translucency (type: scalar):

Scales the shading values. This is the amount of translucency that this shader will apply to the surface the shader is connected to.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

#### Function (type: function):

Can receive inputs from other function nodes in the network only.

Connecting a function node to this input will transform the texture shading value of this shading model based on the graph defined by the function thereby extending the range of this nodes producible shading properties.

See the manual section on Function nodes for a description of how connecting functions can transform the values of the nodes they are connected to.

#### Normal (type: vector):

Normal maps connected to this input will replace the normals that this shader applies shading to in an individually exclusive manner from other shading models. This means that more than one set of normals can be utilized by any shaders exclusively or shared between any of the various shaders and/or the global normals for an individual surface.

The per-surface global normals are defined by either the interpolated geometry itself or by whatever is connected to the





Normal input on the Surface destination node.

By using a different set of normals either from different normal maps or from any of various nodes that may offer normal outputs themselves, it becomes possible to create truly layered surface shading and texturing.

Can receive input from other vector type outputs that define surface normal information. Usually these outputs contain the word "Normal" in the output label text.

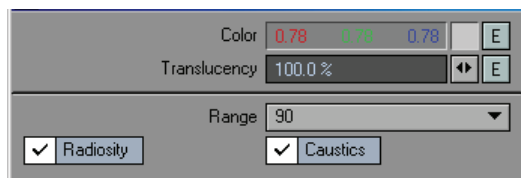
## Outputs

### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

Typically this output is connected to the Diffuse Shading input of the Surface destination node however, it may also be used within a network when using this shading model for other purposes or when mixing several diffuse shading models together.

## Edit Panel

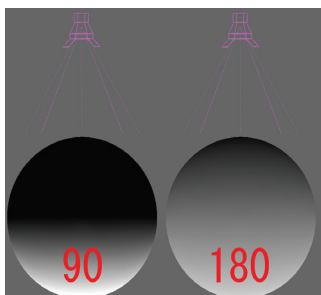


Here the Radiosity and Caustics check boxes as well as the Range settings are not offered as node connections in the Workspace area and are features exclusive to the Edit Panel of the node.

When checked this shading model will evaluate shading information for the surface channel it is being applied including any Radiosity or Caustics computations as defined by those system attributes applied to the scene.

Unchecked, radiosity and/or caustics calculations will not be evaluated by this shading model as it is applied to the object surface being edited.

This can be used to omit both entire surfaces or per channel surface attributes from radiosity or caustics calculations thus offering more diversity in the rendered output as well as reducing the time needed to calculate these surfaces in scenes where either radiosity or caustics are used. The Range pull down menu is also a control exclusive to the edit panel of this node. The options are 90 and 180 and control the maximum angle that the light is allowed to diffuse. Here is a quick visualization to exemplify the affects of this control.



## Shaders (Reflection)

Reflection shaders require that the *amount* of overall surface reflection be set either in the Surface Editor panels' Reflection percentage or by connecting outputs from other nodes in the network to the Reflection input of the surface destination node. Reflection Shading is not the same thing as Reflection amount.

The shading models offered in this category were intended to be used as input to the Reflection Shading connection in the Surface destination node.

The shading models offered in this section can additionally be mixed and modified in order to create a huge variety of different surface shading.

Each significant shading model can shade to an individual set of normals thereby allowing true layered shading per surface.

For example one normal map can be connected to the Lambertian diffuse shader, an entirely different normal map connected to the Phong specular shader and then an additional third unique normal map for the surface globally by connecting it to the Surface destination nodes Normal input. There's no implied limit to the number of shaded Normal layers that can be added to a surface in this way.

This is extremely useful for creating the illusion of depth within a surface. Multiple Normal mapped shading layers can be applied for each attribute; Diffuse, Reflection, Specular, Subsurface Scattering, and Transparent, besides the usual per Surface global normal mapping.



## Ani-Reflections

### Description

Ani-Reflections is a NewTek original adaptation of the Gregory J. Ward anisotropic reflection shading model.

Anisotropy simply put is the property of not being isotropic. An isotropic surface has the property that for any given point on the surface, the light or reflection does not change when the surface is rotated about its normal.

Picture a perfect mirror shaped like a 33.3 rpm record album. Placed on the turntable one would not be able to tell if it were rotating or not by focusing their gaze on any portion of the mirror platter surface. This is isotropic.

Etch a checker pattern of extremely small grooves into the surface however and the same rotation becomes extremely obvious. Depending on what is being reflected the surface may even appear to flash or strobe as the platter rotates. This is anisotropic reflections.

This shading model is very well suited for simulating the multicolored reflections one sees on the surface of a Compact Disk (CD) or Digital Video Disk (DVD).

### Inputs

#### Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color used for the reflectivity channel.

In order to use colored reflection you must first enable Tint Reflections in the Edit Panel for this node.

#### Angle (type: scalar):

Angle or "Image Seam Angle" as it is designated in the Edit Panel for this node allows the user to rotate the seam line of a wrapped image around the Axis that has been selected for the projection method.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

#### Dispersion (type: scalar):

Controls the amount of light dispersion. This input can receive patterns or numerical values from other nodes in the network. This value can additionally be specified via user input by entering values into the Edit Panel for this node.

Dispersion is a phenomenon that causes the separation of a wave into spectral components with different wavelengths, due to a dependence of the waves' speed on its wavelength.

In simpler terms dispersion occurs in materials in which the Index Of Refraction (IOR) is not constant, but rather varies depending on the wavelength of incoming light. White light splits into its monochromatic components, where each wavelength beam has a different IOR. The classic example is a beam of white light entering a prism of some sort and projecting a rainbow colored pattern out the other side.

Dispersion as used in this shading model simulates real light dispersion. The higher the value the greater the amount of dispersion that will be simulated and applied.

#### Samples (type: integer):

Can receive numerical inputs from other nodes in the network. Additionally, controls offered in the Edit Panel for this node may be used to specify this value.

Sampling is the number of directions the shading model examines the surface from in order to calculate the shading value of each spot on the surface under evaluation.

These readings or "Samples" are taken from positions on a hemispherical lattice and it is the number of longitudinal and latitudinal sections of this hemisphere that you are defining when you select one of the Sample preset values from the Samples pull down menu. It is for this reason that there are two numbers in each of the possible Sample selections.

It's quite easy to understand if you imagine one half of an unsmoothed (faceted) default ball object in modeler. The first number defines the number of longitudinal strips and the second number defines the number of latitudinal sections where each facet (or "face") formed by their intersection defines one of the directions from which the spot is evaluated. You could visualize a sample size of 1x4 for example, as a bottomless pyramid centered over the evaluation spot protruding straight out from the surface normal - 1 segment high and containing 4 sides.

Higher values will sample the shading from more directions and produce smoother more accurate shading.

Incoming values higher than 15 will be set to 15. Incoming values lower than 0 will be set to 0.



## U (type: scalar):

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Scales the base of the microfacet pyramids in the U direction.

When a high ratio exists between the U and V scale microfacet pyramids are defined that are much longer than they are wide for example.

Each spot is evaluated with the reflectance of a virtual pyramid pointing up from the surface. The U and V values define the scale of the base of that pyramid.

When U and V are equal the surface reflections will be isotropic. When U and V are unequal anisotropic reflections result.

Smaller U and V values create smoother surface properties. Larger values create rougher looking surfaces.

## V (type: scalar):

Scales the base of the microfacet pyramids in the U direction.

When a high ratio exists between the U and V scale microfacet pyramids are defined that are much longer than they are wide for example.

Each spot is evaluated with the reflectance of a virtual pyramid pointing up from the surface. The U and V values define the scale of the base of that pyramid.

When U and V are equal the surface reflections will be isotropic. When U and V are unequal anisotropic reflections result.

Smaller U and V values create smoother surface properties. Larger values create rougher looking surfaces.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

## Center (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Center receives its X, Y, Z, coordinate information either by entering it into the Center tab in the Edit Panel or by connecting an input to the Center connection on the node in the Workspace Area.

Center defines the center coordinate of the generation point. These are offset values of the objects coordinates also sometimes referred to as the objects pivot point.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Rotation (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Rotation receives H (Heading), P (Pitch), and B (Bank) rotational information either by entering degrees into the Rotation tab in the Edit Panel or as radians by connecting an input to the Rotation connection on the node in the Workspace Area.

Rotation defines the axial rotation around the generation center point. These are offset values of the objects a rotational settings.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Normal (type: vector):

Normal maps connected to this input will replace the normals that this shader applies shading to in an individually exclusive manner from other shading models. This means that more than one set of normals can be utilized by any shaders exclusively or shared between any of the various shaders and/or the global normals for an individual surface.

The per-surface global normals are defined by either the interpolated geometry itself or by whatever is connected to the Normal input on the Surface destination node.

By using a different set of normals either from different normal maps or from any of various nodes that may offer normal outputs themselves, it becomes possible to create truly layered surface shading and texturing.

Can receive input from other vector type outputs that define surface normal information. Usually these outputs contain the word "Normal" in the output label text.

## Outputs

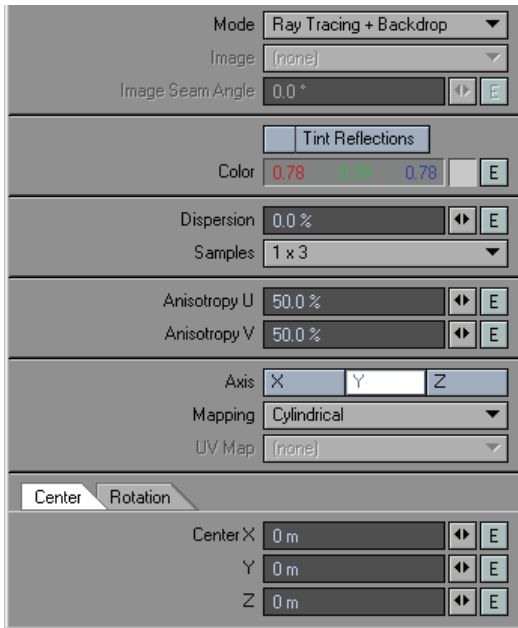
### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

Typically this output is connected to the Reflection Shading input of the Surface destination node. However, it may also be used within a network when using this shading model for other purposes or when mixing several diffuse shading models together.



## Edit Panel



Here the Mapping projection type and Axis are not offered as node connections in the Workspace area. Likewise Mode, Tint Reflections, and Image are features exclusive to this nodes Edit Panel. Please see the explanation for these items in the chapter that covers texturing with the Surface Editor.

Tint Reflection is used in order to enable the Color tinting as described above.

## Reflections

### Description

Reflections is an advanced reflection shading model created by the Finnish Scientist Antti Järvelä. It offers a number of improvements and original features such as wavelength based dispersion, definable sampling, and an intelligent blurring algorithm.

### Inputs

#### Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color used for the reflectivity channel.

#### Angle (type: scalar):

Angle or “Image Seam Angle” as it is designated in the Edit Panel for this node allows the user to rotate the seam line of a wrapped image around the Axis that has been selected for the projection method.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

#### Blur (type: scalar):

Controls the amount of blur of the surface reflections. Some amount of blur is required in order to have the Dispersion amount affect the surface. The number of samples used also greatly affects the quality of blurring that takes place.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

#### Dispersion (type: scalar):

Controls the amount of light dispersion. This input can receive patterns or numerical values from other nodes in the network. This value can additionally be specified via user input by entering values into the Edit Panel for this node.

Dispersion is a phenomenon that causes the separation of a wave into spectral components with different wavelengths, due to a dependence of the waves’ speed on its wavelength.

In simpler terms dispersion occurs in materials in which the Index Of Refraction (IOR) is not constant, but rather varies depending on the wavelength of incoming light. White light splits into its monochromatic components, where each wavelength beam has a different IOR. The classic example is a beam of white light entering a prism of some sort and projecting a rainbow colored pattern out the other side.



Dispersion as used in this shading model simulates real light dispersion. The higher the value the greater the amount of dispersion that will be simulated and applied.

For the affects of Dispersion to be seen there needs to be some amount of Blur set in this node. The amount of Blur controls the relative spread of the Dispersion banding that take place. You can think of this as the width of the rainbow pattern that outlines reflection contrasts in a way.

## Samples (type: integer):

Can receive numerical inputs from other nodes in the network. Additionally, controls offered in the Edit Panel for this node may be used to specify this value.

Sampling is the number of directions the shading model examines the surface from in order to calculate the shading value of each spot on the surface under evaluation.

These readings or "Samples" are taken from positions on a hemispherical lattice and it is the number of longitudinal and latitudinal sections of this hemisphere that you are defining when you select one of the Sample preset values from the Samples pull down menu. It is for this reason that there are two numbers in each of the possible Sample selections.

It's quite easy to understand if you imagine one half of an unsmoothed (faceted) default ball object in modeler. The first number defines the number of longitudinal strips and the second number defines the number of latitudinal sections where each facet (or "face") formed by their intersection defines one of the directions from which the spot is evaluated. You could visualize a sample size of 1x4 for example, as a bottomless pyramid centered over the evaluation spot protruding straight out from the surface normal - 1 segment high and containing 4 sides.

Higher values will sample the shading from more directions and produce smoother more accurate shading.

Incoming values higher than 15 will be set to 15. Incoming values lower than 0 will be set to 0.

## Normal (type: vector):

Normal maps connected to this input will replace the normals that this shader applies shading to in an individually exclusive manner from other shading models. This means that more than one set of normals can be utilized by any shaders exclusively or shared between any of the various shaders and/or the global normals for an individual surface.

The per-surface global normals are defined by either the interpolated geometry itself or by whatever is connected to the Normal input on the Surface destination node.

By using a different set of normals either from different normal maps or from any of various nodes that may offer normal outputs themselves, it becomes possible to create truly layered surface shading and texturing.

Can receive input from other vector type outputs that define surface normal information. Usually these outputs contain the word "Normal" in the output label text.

## Outputs

### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

Typically this output is connected to the Reflection Shading input of the Surface destination node. However, it may also be used within a network when using this shading model for other purposes or when mixing several diffuse shading models together.

## Edit Panel



Here, Tint Reflections is not offered as a node connection in the Workspace area. Likewise Mode and Image are features exclusive to this nodes Edit Panel. Please see the explanation for these items in the chapter that covers texturing with the Surface Editor.

Tint Reflection is used in order to enable the Color tinting as described above.



## Shaders (Specular)

The nodes in this section were designed to apply various kinds of shading to the specular channel of a given named surface. It is useful to keep in mind that Specular shading is not the same thing as the specular amount. Each of the nodes in this section have a Specularity percentage value available in either the node Edit Panel or as a connection input to the node as it appears in the workspace area. It is this percentage value that is responsible for determining the amount of specular shading these shaders will apply to a Surface when connected to the Specular Shading input of the Surface destination node.

The shading models offered in this category were intended to be used as input to the Specular Shading connection in the Surface destination node.

The shading models covered in this section can additionally be mixed and modified in order to create a huge variety of different surface shading. Each significant shading model can shade to an individual set of normals thereby allowing true layered shading per surface.

For example one normal map can be connected to the Lambertian diffuse shader, an entirely different normal map connected to the Phong specular shader and then an additional third unique normal map for the surface globally by connecting it to the Surface destination nodes Normal input. There's no implied limit to the number of shaded Normal layers that can be added to a surface in this way.

This is extremely useful for creating the illusion of depth within a surface. Multiple Normal mapped shading layers can be applied for each attribute; Diffuse, Reflection, Specular, Subsurface Scattering, and Transparent, besides the usual per Surface global normal mapping.

## Anisotropic

### Description

The Anisotropic specular shading model is a NewTek original adaptation of the Gregory J. Ward anisotropic shading model; adapted by the Finnish Scientist Antti Järvelä.

Anisotropy simply put is the property of not being isotropic. An isotropic surface has the property that for any given point on the surface, the reflection of light sources does not change when the surface is rotated about its normal.

Picture a perfect mirror shaped like a 33.3rpm record album. Placed on the turntable one would not be able to tell if it were rotating or not by focusing their gaze on any portion of the mirror platter surface. This is isotropic.

Etch a checker pattern of extremely small grooves into the surface however and the same rotation becomes extremely obvious. Depending on what is being reflected the surface may even appear to flash or strobe as the platter rotates. This is the affects of an anisotropic surface.

This shader is specifically intended for simulating the specular properties of brushed metals such as magnesium automobile rims (mags), various kinds of cookware, and etc. As with most shaders the actual application of the Anisotropic node is limited only by your imagination.

### Inputs

#### Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color used for the specularity channel.

#### Specular (type: scalar):

Defines the *amount* of specularity that is applied to the surface by this node.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.





## U (type: scalar):

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Scales the base of the microfacet pyramids in the U direction.

When a high ratio exists between the U and V scale microfacet pyramids are defined that are much longer than they are wide for example.

Each spot is evaluated with the reflectance of a virtual pyramid pointing up from the surface. The U and V values define the scale of the base of that pyramid.

When U and V are equal the surface reflections will be isotropic. When U and V are unequal anisotropic reflections result.

Smaller U and V values create smoother surface properties. Larger values create rougher looking surfaces.

## V (type: scalar):

Scales the base of the microfacet pyramids in the V direction.

When a high ratio exists between the U and V scale microfacet pyramids are defined that are much longer than they are wide for example.

Each spot is evaluated with the reflectance of a virtual pyramid pointing up from the surface. The U and V values define the scale of the base of that pyramid.

When U and V are equal the surface reflections will be isotropic. When U and V are unequal anisotropic reflections result.

Smaller U and V values create smoother surface properties. Larger values create rougher looking surfaces.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

## Function (type: function):

Can receive inputs from other function nodes in the network only.

Connecting a function node to this input will transform the texture shading value of this shading model based on the graph defined by the function thereby extending the range of this nodes producible shading properties.

See the manual section on Function nodes for a description of how connecting functions can transform the values of the nodes they are connected to.

## Center (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Center receives its X, Y, Z. coordinate information either by entering it into the Center tab in the Edit Panel or by connecting an input to the Center connection on the node in the Workspace Area.

Center defines the center coordinate of the generation point. These are offset values of the objects coordinates also sometimes referred to as the objects pivot point.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Rotation (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Rotation receives H (Heading), P (Pitch), and B (Bank) rotational information either by entering degrees into the Rotation tab in the Edit Panel or as radians by connecting an input to the Rotation connection on the node in the Workspace Area.

Rotation defines the axial rotation around the generation center point. These are offset values of the objects a rotational settings.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Normal (type: vector):

Normal maps connected to this input will replace the normals that this shader applies shading to in an individually exclusive manner from other shading models. This means that more than one set of normals can be utilized by any shaders exclusively or shared between any of the various shaders and/or the global normals for an individual surface.

The per-surface global normals are defined by either the interpolated geometry itself or by whatever is connected to the Normal input on the Surface destination node.

By using a different set of normals either from different normal maps or from any of various nodes that may offer normal outputs themselves, it becomes possible to create truly layered surface shading and texturing.

Can receive input from other vector type outputs that define surface normal information. Usually these outputs contain the word "Normal" in the output label text.



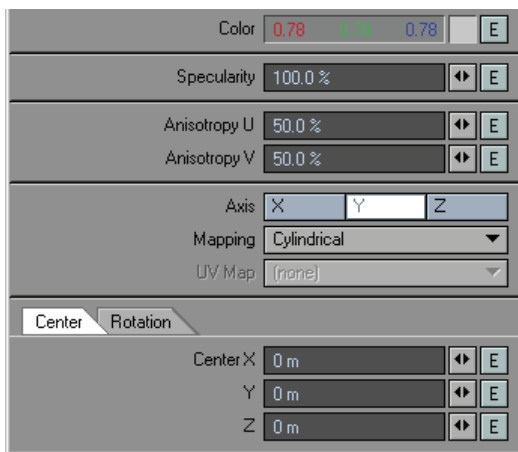
## Outputs

### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

Typically this output is connected to the Specular Shading input of the Surface destination node. However, it may also be used within a network when using this shading model for other purposes or when mixing several diffuse shading models together.

### Edit Panel



Here the Mapping projection type and Axis are not offered as node connections in the Workspace area and are features exclusive to this nodes Edit Panel . Please see the explanation for these items in the chapter that covers texturing with the Surface Editor.

## Blinn

### Description

The Blinn shading model simulates specular reflection. Blinn takes into account changes in specularity due to the angle you are viewing the surface at.

This model produces shading with larger highlights than those occurring in the phong model, especially when illumination strikes the surface at a grazing angle. This also produces soft highlights that tend to be more realistic than the phong model.

Both Phong and Blinn are very commonly found and used in the CG industry. Blinn is the specular model used as default by most OpenGL capable display adapters.

### Inputs

#### Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color used for the specularity channel.

#### Specular (type: scalar):

Defines the *amount* of specularity that is applied to the surface by this node.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

#### Glossiness (type: scalar):

Defines the *amount* of glossiness that is applied to the surface by this node.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

#### Function (type: function):

Can receive inputs from other function nodes in the network only.

Connecting a function node to this input will transform the texture shading value of this shading model based on the graph defined by the function thereby extending the range of this nodes producible shading properties.

See the manual section on Function nodes for a description of how connecting functions can transform the values of the nodes they are connected to.



## Normal (type: vector):

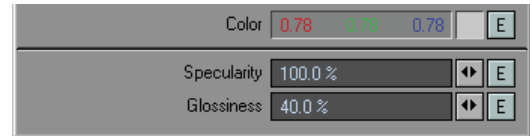
Normal maps connected to this input will replace the normals that this shader applies shading to in an individually exclusive manner from other shading models. This means that more than one set of normals can be utilized by any shaders exclusively or shared between any of the various shaders and/or the global normals for an individual surface.

The per-surface global normals are defined by either the interpolated geometry itself or by whatever is connected to the Normal input on the Surface destination node.

By using a different set of normals either from different normal maps or from any of various nodes that may offer normal outputs themselves, it becomes possible to create truly layered surface shading and texturing.

Can receive input from other vector type outputs that define surface normal information. Usually these outputs contain the word "Normal" in the output label text.

## Edit Panel



## Outputs

### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

Typically this output is connected to the Specular Shading input of the Surface destination node. However, it may also be used within a network when using this shading model for other purposes or when mixing several diffuse shading models together.



## Cook Torrance

### Description

K. E. Torrance and Sparrow designed a physical based BRDF in 1967, that produced a rough surface as a base. The surface used microfacets and the angle of those facets were used to describe surface roughness. It used real world parameters to describe the reflection distribution of wavelength dependent light and therefore a specific point on a given surface could have different colors from different views.

In 1982 R. L. Cook and K. E. Torrance came up with a mixture of the previous shading model and Jim Blinn's model and additionally included more distribution functions for the distribution of the microfacets. The Torrance-Sparrow model used the well known Gauss distribution, Cook-Torrance sometimes also called the Blinn-Cook-Torrance, also included functions for the Phong distribution, the Trowbridge-Reitz-distribution and the Beckmann-distribution.

Together Cook and Torrance also added IOR and angle dependent reflections as well as an geometrical attenuation factor by adding self shadowing and masking calculations to the microfacet idea.

The Cook-Torrance model is a very common BRDF good for simulating a wide variety of materials from metal to some types of cloth.

The OrenNayar (diffuse) model is very common variant of this microfacet usage and model. Cook Torrance specular shading in combination with the OrenNayar diffuse shader can be a much more realistic alternative to the Lambert and Blinn of past LightWave3D versions and have many advantages for both photoreal and NPR rendering.

### Inputs

#### Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color used for the specular channel.

#### Specular (type: scalar):

Defines the *amount* of specular that is applied to the surface by this node.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

#### Glossiness (type: scalar):

Defines the *amount* of glossiness that is applied to the surface by this node.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

#### Function (type: function):

Can receive inputs from other function nodes in the network only.

Connecting a function node to this input will transform the texture shading value of this shading model based on the graph defined by the function thereby extending the range of this nodes producible shading properties.

See the manual section on Function nodes for a description of how connecting functions can transform the values of the nodes they are connected to.

#### Normal (type: vector):

Normal maps connected to this input will replace the normals that this shader applies shading to in an individually exclusive manner from other shading models. This means that more than one set of normals can be utilized by any shaders exclusively or shared between any of the various shaders and/or the global normals for an individual surface.

The per-surface global normals are defined by either the interpolated geometry itself or by whatever is connected to the Normal input on the Surface destination node.

By using a different set of normals either from different normal maps or from any of various nodes that may offer normal outputs themselves, it becomes possible to create truly layered surface shading and texturing.

Can receive input from other vector type outputs that define surface normal information. Usually these outputs contain the word "Normal" in the output label text.

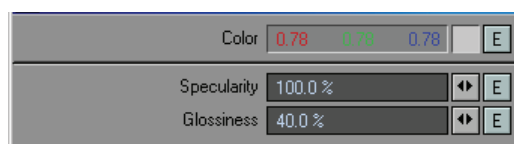
### Outputs

#### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

Typically this output is connected to the Specular Shading input of the Surface destination node. However, it may also be used within a network when using this shading model for other purposes or when mixing several diffuse shading models together.

### Edit Panel





## Phong

### Description

The Phong shading model was invented by Phong Bui Tong in 1975, and because of its simplicity and speed it's one of the most popular models to describe glossy reflections of rendered surfaces.

You can set the parameters so that it reflects more light than it receives, and therefore is not a physically accurate model but because CGI is the *art* of creating illusions this fact can be very useful.

Some of the Phong shaders' tell-tale earmarks are its silky soft hot-spot highlights and its wrap-around rim specular hits gathered from light sources located behind the geometry. This makes it exceptionally useful backlit subjects and characters.

### Inputs

#### Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color used for the specularity channel.

#### Specular (type: scalar):

Defines the *amount* of specularity that is applied to the surface by this node.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

#### Glossiness (type: scalar):

Defines the *amount* of glossiness that is applied to the surface by this node.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

#### Function (type: function):

Can receive inputs from other function nodes in the network only.

Connecting a function node to this input will transform the texture shading value of this shading model based on the graph defined by the function thereby extending the range of this nodes producible shading properties.

See the manual section on Function nodes for a description of how connecting functions can transform the values of the nodes they are connected to.

### Normal (type: vector):

Normal maps connected to this input will replace the normals that this shader applies shading to in an individually exclusive manner from other shading models. This means that more than one set of normals can be utilized by any shaders exclusively or shared between any of the various shaders and/or the global normals for an individual surface.

The per-surface global normals are defined by either the interpolated geometry itself or by whatever is connected to the Normal input on the Surface destination node.

By using a different set of normals either from different normal maps or from any of various nodes that may offer normal outputs themselves, it becomes possible to create truly layered surface shading and texturing.

Can receive input from other vector type outputs that define surface normal information. Usually these outputs contain the word "Normal" in the output labels.

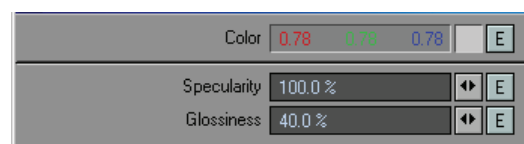
### Outputs

#### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

Typically this output is connected to the Specular Shading input of the Surface destination node. However, it may also be used within a network when using this shading model for other purposes or when mixing several diffuse shading models together.

#### Edit Panel





## Shaders (Subsurface Scattering)

The nodes in this section were designed to apply various kinds of shading to the diffuse channel of a given named surface. It is useful to keep in mind that shading is not the same thing as the amount of SSS that is applied. Each of the nodes in this section have a Amount percentage value available in either the nodes Edit Panel or as a connection input to the node as it appears in the workspace area. It is this percentage value that is responsible for determining the amount of Subsurface Scattering shading these shaders will apply to a surface when connected to the Diffuse Shading input of the Surface destination node.

The shading models offered in this category were intended to be used as input to the Diffuse Shading connection in the Surface destination node.

The shading models covered in this section can additionally be mixed and modified in order to create a huge variety of different surface shading. Each significant shading model can shade to an individual set of normals thereby allowing true layered shading per surface.

For example one normal map can be connected to the Kappa diffuse shader, an entirely different normal map connected to the Phong specular shader and then an additional third unique normal map for the surface globally by connecting it to the Surface destination nodes Normal input. There's no implied limit to the number of shaded Normal layers that can be added to a surface in this way.

This is extremely useful for creating the illusion of depth within a surface. Multiple Normal mapped shading layers can be applied for each attribute; Diffuse, Reflection, Specular, Subsurface Scattering, and Transparent, besides the usual per Surface global normal mapping.

## Kappa

### Description

This nodes' output is intended to be connected to the Diffuse Shading input of the Surface destination node. Other uses may be possible but are not documented and may have unexpected results.

It has the advantage of speed over the "real" SSS model called Omega by the same creator and often looks just as good. Additionally it's easier to set up and does not require raytracing be enabled, Double Sided turned on, nor need a layer of Air conversion polygons be present in the model to work it's magic.

Kappa actually works best and was designed to operate with ray traced shadows turned ON.

The SSS-like results can be dramatically pleasing when soft back-lighting is applied, such as that from an Area Light on the opposite side of the object away from the viewer when the Backward sampling Mode is selected in the Edit Panel for this node.

### Inputs

#### Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color used for the diffuse translucent channel.

#### Range (type: scalar):

Is the maximum range of the sampling or the maximum distance at which samples are considered from the spot being evaluated.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

#### Amount (type: scalar):

Scales the result of the effect which is the output, You can think of this as scaling the amount of SSS that is being applied to the surface.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.





## Samples (type: integer):

Can receive numerical inputs from other nodes in the network. Additionally, controls offered in the Edit Panel for this node may be used to specify this value.

Sampling is the number of directions the shading model examines the surface from in order to calculate the shading value of each spot on the surface under evaluation.

These readings or "Samples" are taken from positions on a hemispherical lattice and it is the number of longitudinal and latitudinal sections of this hemisphere that you are defining when you select one of the Sample preset values from the Samples pull down menu. It is for this reason that there are two numbers in each of the possible Sample selections.

It's quite easy to understand if you imagine one half of an unsmoothed (faceted) default ball object in modeler. The first number defines the number of longitudinal strips and the second number defines the number of latitudinal sections where each facet (or "face") formed by their intersection defines one of the directions from which the spot is evaluated. You could visualize a sample size of 1x4 for example, as a bottomless pyramid centered over the evaluation spot protruding straight out from the surface normal - 1 segment high and containing 4 sides.

Higher values will sample the shading from more directions and produce smoother more accurate shading.

Incoming values higher than 15 will be set to 15. Incoming values lower than 0 will be set to 0.

## Normal (type: vector):

Normal maps connected to this input will replace the normals that this shader applies shading to in an individually exclusive manner from other shading models. This means that more than one set of normals can be utilized by any shaders exclusively or shared between any of the various shaders and/or the global normals for an individual surface.

The per-surface global normals are defined by either the interpolated geometry itself or by whatever is connected to the Normal input on the Surface destination node.

By using a different set of normals either from different normal maps or from any of various nodes that may offer normal outputs themselves, it becomes possible to create truly layered surface shading and texturing.

Can receive input from other vector type outputs that define surface normal information. Usually these outputs contain the word "Normal" in the output labels.

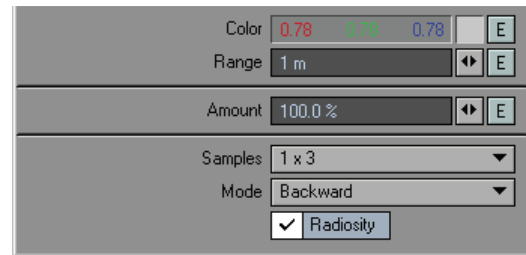
## Outputs

### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

Typically this output is connected to the Diffuse Shading input of the Surface destination node. However, it may also be used within a network when using this shading model for other purposes or when mixing several diffuse shading models together.

## Edit Panel



Here the Radiosity check box is not offered as a node connection in the Workspace area. When checked this shading model will accept properties from the radiosity illumination system.

Unchecked, radiosity calculations will not be applied to this shading model as it is applied to the object surface being edited. This can be used to omit both entire surfaces or per channel surface attributes from radiosity calculations thus offering more diversity in the rendered output as well as reducing the time needed to calculate the surface in scenes where radiosity is used.

Mode determines whether sampling occurs toward the viewer (Forward) or away from the viewer (Backward). You can remember which one to use by their names. Basically Forward Mode is used for when light is entering the material from the front - meaning from a similar direction as the viewer. Backward Mode is suitable for when light is entering from behind the object. Light positioning of all light types matter when setting the Forward or Backward Mode types.

Forward scattering is classically associated with Jade and similar materials. Backward scattering is useful and more predominate in clouds and such.

For realistic human heads and etc, probably both scattering Modes would be needed but with different colors and values. For example Forward scattering with a skin tone coloring and a smaller Range for the skin shading and a reddish colored Backward scattering with deeper or larger, Range for the blood and flesh properties of the model. This can be achieved with multiple instances of the Kappa node mixed, and connected to the Diffuse Shading input of the Surface destination node.



## Kappa II

### Description

This nodes' output is intended to be connected to the Diffuse Shading input of the Surface destination node. Other uses may be possible but are not documented and may have unexpected results.

It has the advantage of speed over the "real" SSS model called Omega by the same creator and often looks just as good. Additionally it's easier to set up and does not require raytracing be enabled, Double Sided turned on, nor need a layer of Air conversion polygons be present in the model to work its magic.

Kappa II actually works best and was designed to operate with ray traced shadows turned ON.

The SSS-like results can be dramatically pleasing when soft back-lighting is applied, such as that from an Area Light on the opposite side of the object away from the viewer when the Backward sampling Mode is selected in the Edit Panel for this node.

### Inputs

#### Forward Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color used for the diffuse translucent channel.

#### Backward Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color used for the diffuse translucent channel.

#### Range (type: scalar):

Is the maximum range of the sampling or the maximum distance at which samples are considered from the spot being evaluated.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

#### Forward Amount (type: scalar):

Scales the result of the effect which is the output, You can think of this as scaling the amount of SSS that is being applied to the surface.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

#### Backward Amount (type: scalar):

Scales the result of the effect which is the output, You can think of this as scaling the amount of SSS that is being applied to the surface.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

#### Samples (type: integer):

Can receive numerical inputs from other nodes in the network. Additionally, controls offered in the Edit Panel for this node may be used to specify this value.

Sampling is the number of directions the shading model examines the surface from in order to calculate the shading value of each spot on the surface under evaluation.

These readings or "Samples" are taken from positions on a hemispherical lattice and it is the number of longitudinal and latitudinal sections of this hemisphere that you are defining when you select one of the Sample preset values from the Samples pull down menu. It is for this reason that there are two numbers in each of the possible Sample selections.

It's quite easy to understand if you imagine one half of an unsmoothed (faceted) default ball object in modeler. The first number defines the number of longitudinal strips and the second number defines the number of latitudinal sections where each facet (or "face") formed by their intersection defines one of the directions from which the spot is evaluated. You could visualize a sample size of 1x4 for example, as a bottomless pyramid centered over the evaluation spot protruding straight out from the surface normal - 1 segment high and containing 4 sides.

Higher values will sample the shading from more directions and produce smoother more accurate shading.

For a table of the possible sampling matrices and the numerical values associated with each, see the end of this document.

#### Normal (type: vector):

Normal maps connected to this input will replace the normals that this shader applies shading to in an individually exclusive manner from other shading models. This means that more than one set of normals can be utilized by any shaders exclusively or shared between any of the various shaders and/or the global normals for an individual surface.

The per-surface global normals are defined by either the interpolated geometry itself or by whatever is connected to the Normal input on the Surface destination node.

By using a different set of normals either from different normal maps or from any of various nodes that may offer normal outputs themselves, it becomes possible to create truly layered surface shading and texturing.

Can receive input from other vector type outputs that define surface normal information. Usually these outputs contain the word "Normal" in the output labels.



## Outputs

### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

Typically this output is connected to the Diffuse Shading input of the Surface destination node. However, it may also be used within a network when using this shading model for other purposes or when mixing several diffuse shading models together.

### Edit Panel

Here the Radiosity check box is not offered as a node connection in the Workspace area. When checked this shading model will accept properties from the radiosity illumination system.

Unchecked, radiosity calculations will not be applied to this shading model as it is applied to the object surface being edited. This can be used to omit both entire surfaces or per channel surface attributes from radiosity calculations thus offering more diversity in the rendered output as well as reducing the time needed to calculate the surface in scenes where radiosity is used.

Mode determines whether sampling occurs toward the viewer (Forward) or away from the viewer (Backward). You can remember which one to use by their names. Basically Forward Mode is used for when light is entering the material from the front - meaning from a similar direction as the viewer. Backward Mode is suitable for when light is entering from behind the object. Positioning of all light types matters when setting the Forward or Backward Mode types.

Forward scattering is classically associated with Jade and similar materials. Backward scattering is useful and more predominate in clouds and such.

For a realistic human head, for example, probably both scattering Modes would be needed but with different colors and values. For example Forward scattering with a skin tone coloring and a smaller Range for the skin shading and a reddish colored Backward scattering with deeper or larger, Range for the blood and flesh properties of the model.

## Omega

### Description

This nodes output is intended to be connected to the Diffuse Shading input of the Surface destination node. Other uses may be possible but are not documented and may have unexpected results.

Subsurface scattering (or SSS) is a mechanism of light transport in which light penetrates the surface of a translucent object, is scattered by interacting with the material, and exits the surface at a different point.

In this shading model light generally penetrates the surface and is reflected a number of times at irregular angles inside the material, before passing back out of the material at an angle other than the angle it would reflect at had it reflected directly off the surface.

Subsurface scattering is an important and recent addition 3D computer graphics for the realistic rendering of such materials as marble, skin, and milk and other real-world semitransparent substances.

Invented by the Finnish Scientist Antti Järvelä in 2006 exclusively for LightWave3D, Omega has the advantage of very highly accurate and versatile. This node requires the RayTraced Shadows be turned on in the Render Globals panel or set in the Render tab under Tracing Options when the Studio Configs are loaded and in use.

Additionally for correct shading this model needs either a layer of flipped conforming polygons set to IOR value of air (between 1.00027419 and 1.00032408) or Double Sided turned on in the Surface Editor and taken advantage of via the Spot Info node by using its Polygon Side output as an input to the Opacity input of a Mixer node connected to the two surfaces. Sometimes the poly layer mentioned above is referred to as "Air Polys".

SSS shading results can be dramatically pleasing when soft back-lighting is applied, such as that from an Area Light on the opposite side of the object away from the viewer when the Backward sampling Mode is selected in the Edit Panel for this node.



## Inputs

### Surface (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

As any particular ray enters the surface it begins as the color defined by Surface. The farther the ray travels into the surface the more it will be tinted by the Subsurface color.

### Subsurface (type: color):

Defines the color that evaluation rays are tinted with the farther they travel into the object.

In other words as evaluation rays start into the surface they are colored 100% of the color defined by the Surface RGB value. The farther the evaluation rays travel into the object the more they are tinted with the Subsurface RGB values.

When a ray reaches the total length of the Penetration multiplied by the Color Falloff value, before exiting then it will return 100% of the Subsurface RGB value upon exit.

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

### Color Falloff (type: scalar):

Color Falloff is the percentage of the Penetration at which the color the ray returns will be 100% of the Subsurface and 0% of the Surface color.

If for example the Penetration were set one meter and the Color Falloff was set to 50% then the Ray would return 100% Subsurface coloring at 50 centimeters and a 50% mix of Surface and Subsurface coloring if the ray exits at 25 centimeters.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

### Falloff Bias (type: scalar):

Falloff Bias applies a curve to the gradient value spread that results from the start color (Surface) to the end color (Subsurface) over the length - which is the Penetration length multiplied by the Color Falloff.

These five parameters (Surface, Subsurface, Color Falloff, Falloff Bias, and Penetration) are actually quite easy to understand by simply visualizing a gradient colored line segment the length of Penetration. At the beginning of the line segment is the start color defined by the RGB values set in Surface. At the end of the gradient spread it the color defined by the RGB values defined in Subsurface. The Color Falloff can bunch up the gradient spread so that the end color of the gradient ends before the end of the line segment or with values higher than 100% can extend the color spread out past the end of the line segment. The Falloff Bias eases in or eases out the spreading of the gradient range values in a nonlinear fashion at values above or below 50% respectively. A value of 50% would produce a linear falloff.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

### Amount (type: scalar):

Scales the output from this node. When connected directly to the Surface destination nodes' Diffuse Shading input for example, it will essentially control the amount of the Omega SSS effect that will be applied to the surface.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

### Spread (type: scalar):

Is the amount of hemispherical spread that evaluation rays are permitted to vary. Higher values will allow evaluation directions at wider angles. Smaller and smaller values will restrict ray direction to a tighter more conically shaped trajectory.

This have the affect of simulating denser or more porous substances.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

### IOR (type: scalar):

Index Of Refraction or IOR, is a charted index of the different amounts of refraction associated with real world materials.

Refraction is the bending of light as it passes through transparent materials or substances of different density.

Dense substances bend light more than materials that are not dense.

See elsewhere in this manual for a list of many real world transparent materials and their corresponding IOR values.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

### Penetration (type: scalar):

Defines the distance at which the color defined by the RGB values set in Surface will be 100% tinted with the RGB values defines by Subsurface given a 100% Color Falloff.

If the Ray Length is equal to or greater that the Penetration value then that ray will return 100% of the Subsurface color.

The overall length of the gradient range may be affected by both the Color Falloff and the Falloff Bias settings.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

### Samples (type: integer):

Can receive numerical inputs from other nodes in the network. Additionally, controls offered in the Edit Panel for this node may be used to specify this value.

Sampling is the number of directions the shading model examines the surface from in order to calculate the shading value of each spot on the surface under evaluation.



These readings or “Samples” are taken from positions on a hemispherical lattice and it is the number of longitudinal and latitudinal sections of this hemisphere that you are defining when you select one of the Sample preset values from the Samples pull down menu. It is for this reason that there are two numbers in each of the possible Sample selections.

It's quite easy to understand if you imagine one half of an unsmoothed (faceted) default ball object in modeler. The first number defines the number of longitudinal strips and the second number defines the number of latitudinal sections where each facet (or “face”) formed by their intersection defines one of the directions from which the spot is evaluated. You could visualize a sample size of 1x4 for example, as a bottomless pyramid centered over the evaluation spot protruding straight out from the surface normal - 1 segment high and containing 4 sides.

Higher values will sample the shading from more directions and produce smoother more accurate shading.

Incoming values higher than 15 will be set to 15. Incoming values lower than 0 will be set to 0.

### Function (type: function):

Can receive inputs from other function nodes in the network only.

Connecting a function node to this input will transform the texture shading value of this shading model based on the graph defined by the function thereby extending the range of this nodes producible shading properties.

See the manual section on Function nodes for a description of how connecting functions can transform the values of the nodes they are connected to.

### Normal (type: vector):

Normal maps connected to this input will replace the normals that this shader applies shading to in an individually exclusive manner from other shading models. This means that more than one set of normals can be utilized by any shaders exclusively or shared between any of the various shaders and/or the global normals for an individual surface.

The per-surface global normals are defined by either the interpolated geometry itself or by whatever is connected to the Normal input on the Surface destination node.

By using a different set of normals either from different normal maps or from any of various nodes that may offer normal outputs themselves, it becomes possible to create truly layered surface shading and texturing.

Can receive input from other vector type outputs that define surface normal information. Usually these outputs contain the word “Normal” in the output labels.

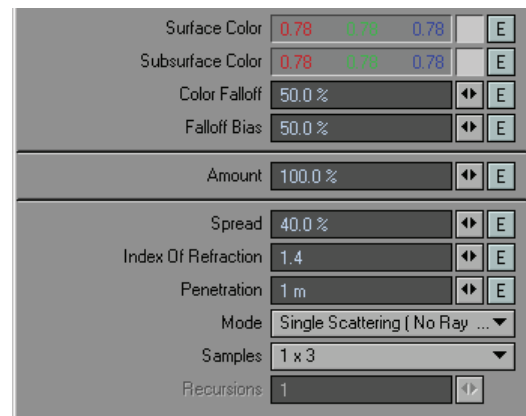
## Outputs

### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

Typically this output is connected to the Diffuse Shading input of the Surface destination node. However, it may also be used within a network when using this shading model for other purposes or when mixing several diffuse shading models together.

### Edit Panel



Here the Mode is not offered as node connections in the Workspace area and is a feature exclusive to this nodes Edit Panel.

Single Scattering (No RayTracing) fires only measurement rays inside the surface in order to measure thickness. This information is then used to calculate the SubSurface scattering as defined in the other properties for this node.

Single Scattering (Full RayTracing) does the same thing but additionally raytraces the color attributes of the ray considering the full scene environment. This enables other objects' surface properties, backdrops, and etc. to be evaluated and affect the overall result of Omega SSS shading.

Multiple Scattering (No RayTracing) is the same as for Single Scattering but additionally allows the rays to bounce around inside the surface collecting multiple sample points. The number of bounces allowed is defined by the Recursions value. Each sample point uses the full Sample matrix defined in the Samples pull-down menu. This results in extremely accurate SSS shading but can also be computationally costly with large values.

Multiple Scattering (Full RayTracing) is the same as for Single Scattering but additionally allows the rays to bounce around inside the surface collecting multiple sample points. The number of bounces allowed is defined by the Recursions value. Each sample point uses the full Sample matrix defined in the Samples pull-down menu. This results in extremely accurate SSS shading but can also be computationally costly with large values.

The Recursion value available when either of the Multiple Scattering options are selected will usually never need to be set any higher than 4 or 5.





## Shaders (Transparency)

The nodes in this section were designed to apply various kinds of shading to the transparency channel of a given named surface. It is useful to keep in mind that shading is not the same thing as the amount of transparency that is applied.

These nodes depend on the Transparency amount set in the Surface Editor panels' Basic tab. It is this percentage value that is responsible for determining the amount of Transparency shading these shaders will apply to a surface when connected to the Refraction Shading input of the Surface destination node.

The shading models offered in this category were intended to be used as input to the Refraction Shading connection in the Surface destination node.

The shading models covered in this section can additionally be mixed and modified in order to create a huge variety of different surface shading. Each significant shading model can shade to an individual set of normals thereby allowing true layered shading per surface.

For example one normal map can be connected to the Kappa diffuse shader, an entirely different normal map connected to the Phong specular shader another unique normal map for Normal input of the Refractions node and then an additional fourth unique normal map for the surface globally by connecting it to the Surface destination nodes Normal input. There's no implied limit to the number of shaded Normal layers that can be added to a surface in this way.

This is extremely useful for creating the illusion of depth within a surface. Multiple Normal mapped shading layers can be applied for each attribute; Diffuse, Reflection, Specular, Subsurface Scattering, and Transparent, besides the usual per Surface global normal mapping.

## Ani-Refractions

### Description

The Ani-Refractions shading model is a NewTek original adaptation of the Gregory J. Ward anisotropic shading model; adapted by the Finnish Scientist Antti Järvelä.

Anisotropy simply put is the property of not being isotropic. An isotropic surface has the property that for any given point on a surface, the reflection or in this case refraction, of light sources does not change when the surface is rotated about its normal.

Picture a perfect mirror shaped like a 33.3rpm record album. Placed on the turntable one would not be able to tell if it were rotating or not by focusing their gaze on any portion of the mirror platter surface. This is isotropic.

Etch a checker pattern of extremely small grooves into the surface however and the same rotation becomes extremely obvious. Depending on what is being reflected the surface may even appear to flash or strobe as the platter rotates. This is the affects of an anisotropic surface.

This shader is specifically intended for simulating the refractive properties of anisotropic transparent materials such as some types of Cut-Glass, Calcite Crystal, and etc.

As with most shaders the actual application of the Ani-Refractions node is limited only by your imagination.

### Inputs

#### Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Specifies the color used for the transparency channel.

#### Angle (type: scalar):

Angle or "Image Seam Angle" as it is designated in the Edit Panel for this node allows the user to rotate the seam line of a wrapped image around the Axis that has been selected for the projection method.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.





## IOR (type: scalar):

Index Of Refraction or IOR, is a charted index of the different amounts of refraction associated with real world materials.

Refraction is the bending of light as it passes through transparent materials or substances of different density.

Dense substances bend light more than materials that are not dense.

See elsewhere in this manual for a list of many real world transparent materials and their corresponding IOR values.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

## Dispersion (type: scalar):

Controls the amount of light dispersion. This input can receive patterns or numerical values from other nodes in the network. This value can additionally be specified via user input by entering values into the Edit Panel for this node.

Dispersion is a phenomenon that causes the separation of a wave into spectral components with different wavelengths, due to a dependence of the waves' speed on its wavelength.

In simpler terms dispersion occurs in materials in which the Index Of Refraction (IOR) is not constant, but rather varies depending on the wavelength of incoming light. White light splits into its monochromatic components, where each wavelength beam has a different IOR. The classic example is a beam of white light entering a prism of some sort and projecting a rainbow colored pattern out the other side.

Dispersion as used in this shading model simulates real light dispersion. The higher the value the greater the amount of dispersion that will be simulated and applied.

## Samples (type: integer):

Can receive numerical inputs from other nodes in the network. Additionally, controls offered in the Edit Panel for this node may be used to specify this value.

Sampling is the number of directions the shading model examines the surface from in order to calculate the shading value of each spot on the surface under evaluation.

These readings or "Samples" are taken from positions on a hemispherical lattice and it is the number of longitudinal and latitudinal sections of this hemisphere that you are defining when you select one of the Sample preset values from the Samples pull down menu. It is for this reason that there are two numbers in each of the possible Sample selections.

It's quite easy to understand if you imagine one half of an unsmoothed (faceted) default ball object in modeler. The first number defines the number of longitudinal strips and the second number defines the number of latitudinal sections where each facet (or "face") formed by their intersection defines one of the directions from which the spot is evaluated. You could visualize a sample size of 1x4 for example, as a bottomless pyramid centered over the evaluation spot protruding straight out from the surface normal - 1 segment high and containing 4 sides.

Higher values will sample the shading from more directions and produce smoother more accurate shading.

Incoming values higher than 15 will be set to 15. Incoming values lower than 0 will be set to 0.

## U (type: scalar):

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Scales the base of the microfacet pyramids in the U direction.

When a high ratio exists between the U and V scale microfacet pyramids are defined that are much longer than they are wide for example.

Each spot is evaluated with the refractions from a virtual pyramid pointing up from the surface. The U and V values define the scale of the base of that pyramid.

When U and V are equal the surface refraction will be isotropic. When U and V are unequal anisotropic refractions result.

Smaller U and V values create smoother surface properties. Larger values create rougher looking surfaces.

## V (type: scalar):

Scales the base of the microfacet pyramids in the V direction.

When a high ratio exists between the U and V scale microfacet pyramids are defined that are much longer than they are wide for example.

Each spot is evaluated with the reflectance of a virtual pyramid pointing up from the surface. The U and V values define the scale of the base of that pyramid.

When U and V are equal the surface reflections will be isotropic. When U and V are unequal anisotropic reflections result.

Smaller U and V values create smoother surface properties. Larger values create rougher looking surfaces.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.



## Center (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Center receives its X, Y, Z, coordinate information either by entering it into the Center tab in the Edit Panel or by connecting an input to the Center connection on the node in the Workspace Area.

Center defines the center coordinate of the generation point. These are offset values of the objects coordinates also sometimes referred to as the objects pivot point.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Rotation (type: vector):

Can receive patterns or numerical values from other nodes in the network.

Rotation receives H (Heading), P (Pitch), and B (Bank) rotational information either by entering degrees into the Rotation tab in the Edit Panel or as radians by connecting an input to the Rotation connection on the node in the Workspace Area.

Rotation defines the axial rotation around the generation center point. These are offset values of the objects a rotational settings.

Values entered into the edit panel via user input will be overridden by any connections made to the node in the workspace area.

## Normal (type: vector):

Normal maps connected to this input will replace the normals that this shader applies shading to in an individually exclusive manner from other shading models. This means that more than one set of normals can be utilized by any shaders exclusively or shared between any of the various shaders and/or the global normals for an individual surface.

The per-surface global normals are defined by either the interpolated geometry itself or by whatever is connected to the Normal input on the Surface destination node.

By using a different set of normals either from different normal maps or from any of various nodes that may offer normal outputs themselves, it becomes possible to create truly layered surface shading and texturing.

Can receive input from other vector type outputs that define surface normal information. Usually these outputs contain the word "Normal" in the output labels.

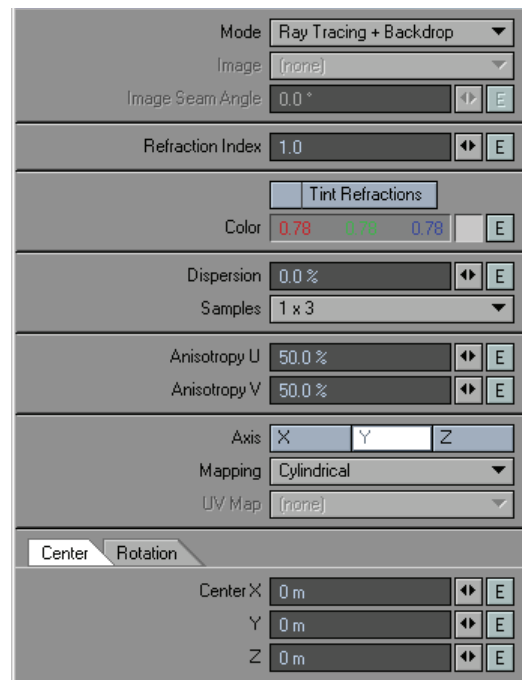
## Outputs

### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

Typically this output is connected to the Reflection Shading input of the Surface destination node. However, it may also be used within a network when using this shading model for other purposes or when mixing several diffuse shading models together.

## Edit Panel



Here the Mapping projection type and Axis are not offered as node connections in the Workspace area. Likewise Mode, Tint Refractions, and Image are features exclusive to this nodes Edit Panel. Please see the explanation for these items in the chapter that covers texturing with the Surface Editor.

Tint Refractions is used in order to enable the Color tinting as described above.



## Refractions

### Description

Refractions is an advanced transparency shading model created by the Finnish Scientist Antti Järvelä exclusively for NewTek's LightWave3D application. It offers a number of improvements and original features such as wavelength based dispersion, definable sampling, and an intelligent blurring algorithm.

### Inputs

#### Color (type: color):

Can receive colors and patterns from other nodes in the network. Users may also specify a color using the controls found in the Edit Panel for this node.

Requires that Tint Refractions be checked on in the Edit Panel for this node.

Specifies the color used for the transparency channel.

#### Angle (type: scalar):

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

Angle or "Image Seam Angle" as it is designated in the Edit Panel for this node allows the user to rotate the seam line of a wrapped image around the Axis that has been selected for the projection method.

#### IOR (type: scalar):

Index Of Refraction or IOR, is a charted index of the different amounts of refraction associated with real world materials.

Refraction is the bending of light as it passes through transparent materials or substances of different density.

Dense substances bend light more than materials that are not dense.

See elsewhere in this manual for a list of many real world transparent materials and their corresponding IOR values.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

#### Blur (type: scalar):

Controls the amount of blur of the surface refractions. Some amount of blur is required in order to have the Dispersion amount affect the surface. The number of samples used also greatly affects the quality of blurring that takes place.

Can receive patterns or numerical values from other nodes in the network. This value can additionally be specified by user input by entering values into the Edit Panel of this node.

#### Dispersion (type: scalar):

Controls the amount of light dispersion. This input can receive patterns or numerical values from other nodes in the network. This value can additionally be specified via user input by entering values into the Edit Panel for this node.

Dispersion is a phenomenon that causes the separation of a wave into spectral components with different wavelengths, due to a dependence of the waves' speed on its wavelength.

In simpler terms dispersion occurs in materials in which the Index Of Refraction (IOR) is not constant, but rather varies depending on the wavelength of incoming light. White light splits into its monochromatic components, where each wavelength beam has a different IOR. The classic example is a beam of white light entering a prism of some sort and projecting a rainbow colored pattern out the other side.

Dispersion as used in this shading model simulates real light dispersion. The higher the value the greater the amount of dispersion that will be simulated and applied.

#### Samples (type: integer):

Can receive numerical inputs from other nodes in the network. Additionally, controls offered in the Edit Panel for this node may be used to specify this value.

Sampling is the number of directions the shading model examines the surface from in order to calculate the shading value of each spot on the surface under evaluation.

These readings or "Samples" are taken from positions on a hemispherical lattice and it is the number of longitudinal and latitudinal sections of this hemisphere that you are defining when you select one of the Sample preset values from the Samples pull down menu. It is for this reason that there are two numbers in each of the possible Sample selections.

It's quite easy to understand if you imagine one half of an unsmoothed (faceted) default ball object in modeler. The first number defines the number of longitudinal strips and the second number defines the number of latitudinal sections where each facet (or "face") formed by their intersection defines one of the directions from which the spot is evaluated. You could visualize a sample size of 1x4 for example, as a bottomless pyramid centered over the evaluation spot protruding straight out from the surface normal - 1 segment high and containing 4 sides.

Higher values will sample the shading from more directions and produce smoother more accurate shading.

Incoming values higher than 15 will be set to 15. Incoming values lower than 0 will be set to 0.



## Normal (type: vector):

Normal maps connected to this input will replace the normals that this shader applies shading to in an individually exclusive manner from other shading models. This means that more than one set of normals can be utilized by any shaders exclusively or shared between any of the various shaders and/or the global normals for an individual surface.

The per-surface global normals are defined by either the interpolated geometry itself or by whatever is connected to the Normal input on the Surface destination node.

By using a different set of normals either from different normal maps or from any of various nodes that may offer normal outputs themselves, it becomes possible to create truly layered surface shading and texturing.

Can receive input from other vector type outputs that define surface normal information. Usually these outputs contain the word "Normal" in the output labels.

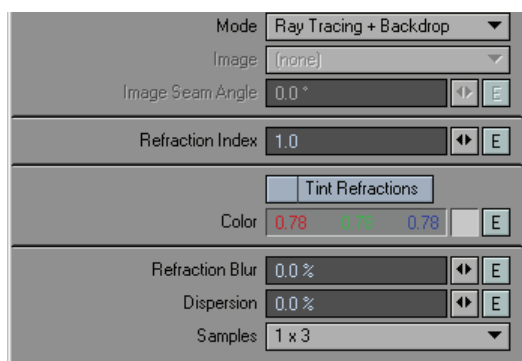
## Outputs

### Color (type: color):

Outputs three channel color information in R, G, B format, evaluated on a per spot basis.

Typically this output is connected to the Reflection Shading input of the Surface destination node. However, it may also be used within a network when using this shading model for other purposes or when mixing several diffuse shading models together.

## Edit Panel



Here, Tint Refractions is not offered as a node connection in the Workspace area. Likewise Mode and Image are features exclusive to this nodes Edit Panel. Please see the explanation for these items in the chapter that covers texturing with the Surface Editor.

## Spot

## Spot Info

### Purpose

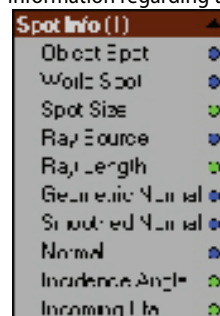
The Spot Info node allows access to all information regarding the current hit point.

### Description

First let's define what a "spot" is. A spot is just a spot - a fleck, a pinpoint. You can think of a spot as a single element area that is calculated or evaluated, on a surface and that is arbitrary in size. Any given spot contains several types of data including color, normal angle, the direction that a ray is coming or going from relative to that spot, and etc.

Remember Node Editor is a surface texture and shading editor so the word spot here pertains to the spots of an objects surface. Spots are a useful primitive type for texture design, because, in general, the relations between features of the spot and features of the texture are straightforward.

The Spot Info is a stand-alone node that provides access to information regarding the current hit point being evaluated.



### Inputs

None

### Outputs

#### Object Spot (Type: Vector):

The Object Spot is the spot being evaluated in object coordinates X, Y, Z.

#### World Spot (Type: Vector):

World Spot is the spot being evaluated in world coordinates X, Y, Z.

#### Spot size (Type: Scalar):

Spot size is the size of the spot currently being evaluated output as a scalar.



## Tools

### Ray source (type: vector):

Ray source is the X, Y, Z coordinate position of the ray source being evaluated.

### Ray length (Type: Scalar):

Ray length is the length of the ray from the source to the spot under evaluation.

### Geometric Normal (Type: Vector):

The Geometric Normal is the normal for the flat-shaded polygon under the current spot of evaluation.

### Smoothed Normal (Type: Vector):

The Smoothed Normal is the normal information for a smooth-shaded polygon under the current spot of evaluation.

### Normal (Type: Vector):

The Normal is the normal for smooth, bump-mapped, shaded polygons under the current spot of evaluation.

### Incidence Angle (Type: Scalar):

Incidence angle is the angle between the source evaluation ray and the normal.

### Incoming ETA (Type: Scalar):

Incoming ETA or Incoming Refraction Index. ETA is the name of the Greek letter commonly used in 3D graphics to specify refraction Index. When the Greek character eta is unavailable, the letter "n" is often used to represent the refractive index in many online discussions and examples.

### Edit Panel

None.

## Color Scalar

### Purpose

The Color Scalar node allows a color to be converted to a proportional scalar.

### Description

The Color Scalar node allows a color to be converted into a scalar. There are several available modes that determine how the conversion occurs. These are:

- Average, which outputs the scalar as the mean of the three color components.

$$\text{Scalar} = (\text{Red} + \text{Green} + \text{Blue}) / 3.0$$

- Maximum, which outputs the scalar as the largest of the three color components.

$$\text{Scalar} = \text{Max}(\text{Red}, \text{Green}, \text{Blue})$$

- Minimum, which outputs the scalar as the smallest of the three color components.

$$\text{Scalar} = \text{Min}(\text{Red}, \text{Green}, \text{Blue})$$

- Red Channel, which outputs the scalar as the value of the red component.

$$\text{Scalar} = \text{Red}$$

- Green Channel, which outputs the scalar as the value of the green component.

$$\text{Scalar} = \text{Green}$$

- Blue Channel, which outputs the scalar as the value of the blue component.

$$\text{Scalar} = \text{Blue}$$

- Luma, which outputs the scalar as the luminance of the input color.

$$\text{Scalar} = \text{CCIR 601 luminance values formula } Y' = 0.299 R' + 0.587 G' + 0.114 B' \text{ so } (\text{Red} \times 0.299) + (\text{Green} \times 0.587) + (\text{Blue} \times 0.114)$$



### Inputs

#### Color (Type: Color):

This input is the color to be converted into a scalar.

#### Mode (Selection: Conversion Mode)

This allows the conversion mode, described above, to be selected.

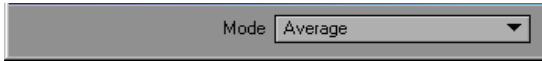


## Outputs

### Scalar (Type: Scalar):

The output is driven with the color converted to a scalar as per the selected conversion mode.

### Edit Panel



## ColorTool

### Purpose

The ColorTool node allows a color to be manipulated using a variety of the controls.

### Description

The Color Tool node takes the input color and provides a set of controls that allow it to be manipulated or processed and then output. The available manipulations are:

- Hue control, which allows the hue of the input color to be shifted by any number of degrees. The HSV color model specifies H as Hue being an angle with red at zero degrees, green at 120 degrees, blue at 240 degrees, and so on. The input color is converted to HSV and the derived hue shifted by the specified amount.
- Saturation control, which works in much the same manner as Hue. Again, as with the HSV color model, S is the saturation where 0% is the absence of any hue (grey scale) and 100% is full color saturation. The input color is then converted to HSV. This control is used to specify the saturation. This has the effect of being able to fade colors to grey.
- Brightness control, which allows the overall brightness of the color to be controlled. Again, this is based on the HSV color model and here it is the V, or value, that is specified by this control. This control works in the same way of the brightness control on a television.
- Contrast control, which allows the contrast of the input color to be altered. This control works in the same way of the contrast control on a television.







## Inputs

### Color (Type: Color):

This input is the color to be processed.

### Hue (Type: Scalar):

This input allows the hue of the input color to be rotated.

### Saturation (Type: Scalar):

This input allows the saturation of the input color to be adjusted in the same way as one might adjust the color control on a television.

### Brightness (Type: Scalar):

This input allows the brightness of the input color to be adjusted in the same way as one might adjust the brightness control on a television.

### Contrast (Type: Scalar):

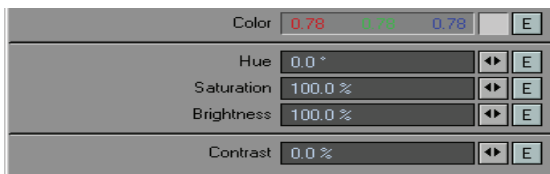
This input allows the contrast of the input color to be adjusted in the same way as one might adjust the contrast on a television.

## Outputs

### Color (Type: Color):

The output is driven with the post-processed color.

## Edit Panel



## Limiter

## Purpose

The Limiter node allows each channel of a color to be clamped between two values.

## Description

The Limiter node clamps each channel of the input color between the user specified low and high limits. Therefore, none of the Red, Green, or Blue components of the input color will be output as being lower than the value specified by the Low input. Conversely, none of the Red, Green, or Blue components of the input color will be output as being higher than the value specified by the High input. Imagine three Clamp nodes each with the same limits placed on the red, green, and blue components of the color.



## Inputs

### Color (Type: Color):

This input is the color to be processed.

### Low (Type: Scalar):

This input specifies the lowest value for any of the components in the input color. If any are below this value, they will be clamped.

### High (Type: Scalar):

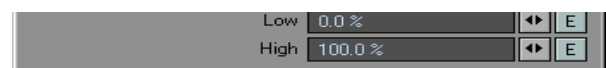
This input specifies the highest values for any of the components in the input color. If any are above this value, they will be clamped.

## Outputs

### Color (Type: Color):

The output is driven with the clamped, or limited, input color.

## Edit Panel





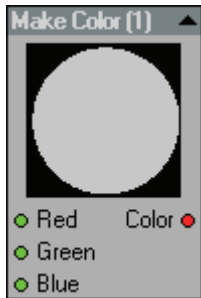
## Make Color

### Purpose

The Make Color node allows a color to be created from three input scalars.

### Description

The Make Color node allows a color to be constructed using three scalars. The scalars used are input on the Red, Green, and Blue inputs, respectively, and must each vary between 0.0 and 1.0.



### Inputs

**Red** (Type: Scalar):

This input is used as the red component of the output color.

**Green** (Type: Scalar):

This input is used as the green component of the output color.

**Blue** (Type: Scalar):

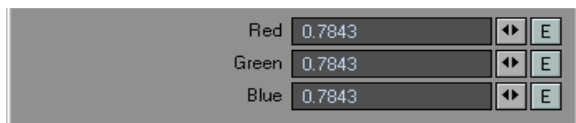
This input is used as the blue component of the output color.

### Outputs

**Color** (Type: Color):

The output is driven with the color constructed using the Red, Green, and Blue inputs from the RGB channels.

### Edit Panel



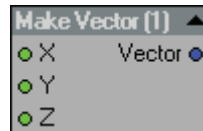
## Make Vector

### Purpose

The Make Vector node allows a vector to be created from three scalars.

### Description

The Make Vector node allows a vector to be constructed using three scalars. The scalars are input on the X, Y, and Z inputs, respectively.



### Inputs

**X** (Type: Scalar); **Y** (Type: Scalar); **Z** (Type: Scalar):

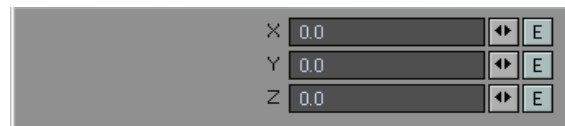
These inputs make up the X, Y, and Z of the output vector.

### Outputs

**Color** (Type: Vector):

The output is driven with vector constructed using the X, Y, and Z inputs.

### Edit Panel





## Mixer

### Purpose

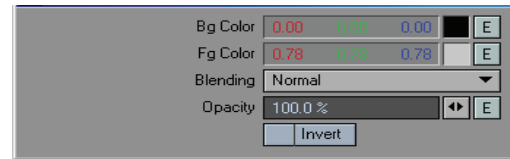
The Mixer node allows two colors to be mixed based on a scalar input.

### Description

The Mixer node allows two colors to be mixed based on a scalar using the standard nodal blending modes. In other words, it can be used to composite two colors. The Fg Color is mixed with the Bg Color using the Opacity as the alpha component for the mix. The two colors will be mixed based on the selected blending mode. For more information about the blending modes see reference section of blending. The mixed, or blended, color will be output on the Color output, while the Alpha output is driven with the value used for the opacity input.



### Edit Panel



### Inputs

**Fg Color** (Type: Color);, **Bg Color** (Type: Color):

These are the two colors to be mixed.

**Blending** (Type: Integer):

This input is used to select the required blending mode.

**Opacity** (Type: Scalar):

This input defines the amount of mixing between the Bg and Fg colors.

**Invert** (Type: Selection):

This input allows opacity input to be inverted prior to being used for the color mix. This allows a mix going Bg to Fg to be swapped so that it goes Fg to Bg.

### Outputs

**Color** (Type: Color):

The output is driven with the result of the Bg and Fg color mix.

**Alpha** (Type: Scalar):

The output is driven with the value used for the Opacity input.



## Vector Scalar

### Purpose

The Vector Scalar allows a vector to be converted to a proportional scalar.

### Description

The Vector Scalar node allows a vector to be converted into a scalar. There are several available modes that determine how the conversion occurs. These are:

- Maximum, which outputs the scalar as the largest of the three vector input components.

Scalar = Max(X, Y, Z)

- Minimum, which outputs the scalar as the smallest of the three vector input components.

Scalar = Min(X, Y, Z)

- X Channel, which outputs the value of the X component of the input vector.

Scalar = X

- Y Channel, which outputs the value of the Y component of the input vector.

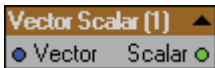
Scalar = Y

- Z Channel, which outputs the value of the Z component of the input vector.

Scalar = Z

- Length, which outputs the length of the input vector.

Scalar = Length(X, Y, Z)



### Inputs

**Vector** (Type: Vector):

This input is the vector to be converted into a scalar.

**Mode** (Selection: Conversion Mode)

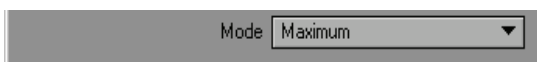
This allows the conversion mode, described above, to be selected.

### Outputs

**Scalar** (Type: Scalar):

This output is driven with the vector converted to a scalar as per the selected conversion mode.

### Edit Panel



## Vertex Map

### Morph Map

### Purpose

The Morph Map Vertex Map node allows access to any current morph maps assigned to the object being shaded.

### Description

The Morph Map node allows access to any morph maps assigned to an object (relative or absolute). Once selected, the Position output will be driven with the morph amount derived for the current hit point and the selected morph map. This allows the amount of morphing to be considered in any shading or texture networks such as making the surface color change proportionally.



### Inputs

**Type** (Selection: Relative or Absolute):

Select whether the morph maps to be considered are relative or absolute.

**Vertex Map** (Selection: Any of the Current Object's Morph Maps):

This drop-down list is filled with all the morph maps assigned to the current object. Any of these can then be selected and used for the derivation of the Position output.

**Amount** (Type: Scalar):

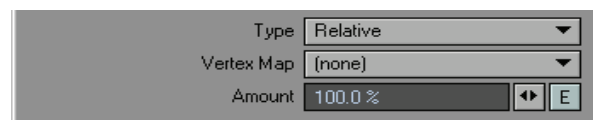
The Amount input allows the size of the direction vector to be scaled. The scaling is based on a percentage such that 100% is full scale, 0% is half scale and -100% is scaled to nothing.

### Outputs

**Position** (Type: Vector):

This vector provides the amount of morph based on the morph map.

### Edit Panel





## Vertex Map

### Purpose

The Vertex Map node allows access to any vertex maps (i.e. color maps) assigned to the object being shaded.

### Description

The Vertex Map node allows access to any color maps assigned to an object. Once selected, the Color output will be driven with the color of the selected vertex map. As with all the color based nodes, the vertex map color can be mixed with the Bg Color. This mixing is based on the currently selected blending mode and the alpha channel derived for the color map. For more information on how the vertex map color and Bg Color are mixed based on the blending mode see reference the blending mode description

When the vertex map is just an RGB map, it has a constant alpha value.



### Inputs

**Type** (Selection: RGB or RGBA):

Select whether the vertex maps to be considered are RGB (just color) or RGBA (Color plus alpha).

**Vertex Map** (Selection: Any of the Current Object's Vertex Maps):

This drop-down list is filled with all the morph maps assigned to the current object. Any of these can then be selected and used for the derivation of the Direction output.

**Amount** (Type: Scalar):

The Amount input allows the size of the direction vector to be scaled. The scaling is based on a percentage such that 100% is full scale, 0% is half scale, and -100% is scaled to nothing.

### Outputs

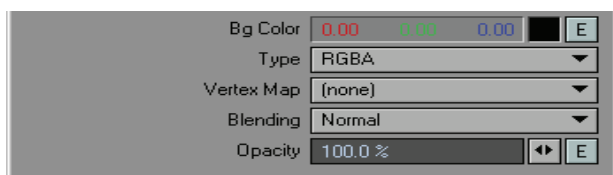
**Color** (Type: Vector):

The color output is driven with the color derived from the selected vertex map for the current hit point being evaluated.

**Alpha** (Type: Scale):

The Alpha output is driven with the alpha derived from the selected vertex map for the current hit point being evaluated.

### Edit Panel



## Weight Map

### Purpose

The Weight Map Vertex node allows access to any weight maps assigned to the object being shaded.

### Description

The Weight Map node allows access to any weight maps assigned to the object being shaded. Once selected, the Value output will be driven with the morph amount derived for the current hit point and the selected weight map.



### Inputs

**Weight Map** (Selection: Any of the Current Object's Weight Maps):

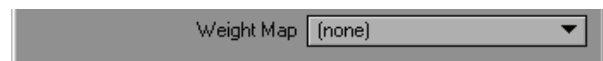
This drop-down list is filled with all the weight maps assigned to the current object. Any of these can then be selected and used for the derivation of the Value output.

### Output

**Value** (Type: Scalar):

This scalar provides the weight based on the selected weight map.

### Edit Panel





## Nodal Glossary

### 2D textures

Images that are mapped to the object surface during the shading computation. An algorithm is used for “wrapping” the texture around the object’s surface so that the pattern curves and distorts realistically. The advantage of using texture is that apparent details do not have to be modeled in geometry. A common use of texture is to apply a label as a texture image on a three-dimensional bottle. 2D textures can also be used as transparency, reflection, and displacement maps.

### 3D textures

A 3D texture is computer generated via procedural algorithm with a set of parameters instead of a picture file. The generated pattern follows logically inside the procedural volume and wherever polygonal surfaces exist in relation to that volume the procedural pattern information is applied. 3D textures can exist in world space coordinates or tied to an object’s center point when world coordinates is unchecked. 3D textures avoid many of the difficulties attributed to mapping a bitmap around the surface of an irregular object. These textures have no edges and provide a continuous-looking appearance.

### Blinn shading

A James Blinn developed method of computing the shading of three-dimensional surfaces. It uses four characteristics: diffusion, specularity, eccentricity, and refractive index. It is very similar to Phong shading, except that it creates slightly different highlights that are useful for defining rough edges or sharp edges. See also Cook-Torrance, Lambert, and Phong shading.

### Bump mapping

The process of applying a texture where the values of the texture are used to simulate the bumpiness of an object’s surface. Bump mapping is used to add detail to an image without increasing the geometric density (number of polygons). Bump maps do not move the geometry of an object, but they do affect the shading of a surface, producing the illusion of the texture pattern being etched or embossed on the surface. Lighter grayscale pixels (higher texture values) produce the maximum amount of perceived relief or maximum indentation; darker pixels (lower values) have less effect. See also Displacement mapping.

### Caustics

A light pattern created by specular reflection or refraction of light, such as the patterns of light on the bottom of a swimming pool or as seen when light passes through a glass of water placed on a flat surface such as a table.

### Color

A data type consisting of red, green, blue, and optional alpha components.

### Cook-Torrance shading

A shading model similar to Blinn, it reads the surface normal

orientations and interpolates between them to create smoothed shading. It also processes the relation among normals, scene lighting, and the camera’s view angle in order to create a specular highlight. This shading model is useful for simulating smooth and reflective objects, such as Naugahyde or vinyl, etc.. Reflectivity, transparency, refraction, and texture can also be applied to a surface shaded by Cook-Torrance. See also Blinn, Lambert, and Phong shading.

### Diffuse

The essential color of an object. This is the color that the object reveals under pure white light. It is the color of the object itself rather than the perceived color produced by the reflection of light. Diffuse color is distinguished from the color of specular reflection (color highlights) from a surface, which are typically the color of the light source itself. See also Specular.

### Displacement mapping

A technique where by the vertices of an object are moved or “displaced” so that the object’s geometry is altered to create a bumpy or deformed surface. Unlike regular bump mapping, the geometrys’ vertices are actually raised or lowered and can cast shadows. The contrasting values of a 2D texture are used to adjust the degree to which the object’s geometry is displaced. With displacement mapping you can create highly complex objects without having to actually model them. See also Bump mapping.

### Lambert shading

A shading model based on the application of Lambert’s cosine law, which deals with the intensity of reflected light, discovered in the 16th century by Johann Lambert. Object surfaces are shaded to create a matte surface with no specular highlights. Two illumination areas are defined on the object’s surface: ambient and diffuse. Lambert shading allows reflectivity, transparency, refraction, textures, and etc. to also be applied to the surface. See also Blinn, Cook-Torrance, and Phong shading.

### Local coordinates

An object’s coordinate system with its own center as the reference point (origin) as compared to world coordinates, which includes an origin for all objects in world space. For example, if you animate a child object in a hierarchy using local coordinates, the motion path is relative to the parent item and not the global origin.

### Mapping

The process of making one image conform to the size, shape, and/or texture of another image. See also Bump mapping, Displacement mapping. This is also the process of applying vertex maps to a group of points on an object.

### Node

Any type of data that is represented in a network layout view in the Workspace area. Each node is identified by its name. The Node Editor Workspace area and Scrollable Node List both, display nodes.

### Normals

Vectors at right angles to the plane or surface of an object. An object’s normals are represented by thin dashed user-definable-colored lines in LightWave Modeler. Normals indicate the side of the object that should be visible to the camera. By default, the renderer shows only the side with the normals.





## Parameters

Also known generally as properties, parameters are the “atomic” elements of a property set whose values determine the behavior of something. You can set parameters manually in a node’s Edit Panel or allow parameters to be controlled by other nodes by connecting a node’s output to the input for that parameter.

## Phong shading

Phong reflection is a local illumination model devised by Bui Tuong Phong used for assigning shades to each individual pixel of 3D object surfaces. The Phong model combines three elements - diffuse, specular and ambient for each considered spot on a surface in order to shade curved surfaces with light-reflecting highlights. See also Blinn, Cook-Torrance, and Lambert shading.

## Procedural textures

See 3D textures.

## Raytracing

A rendering method that plots a view of every pixel in a scene through LightWave’s virtual camera lens. It works by tracing the path taken by a ray of light cast from the camera through the scene, then calculating the reflection, refraction, or absorption of the ray when it intersects objects, and other scene elements in the scene taking into account the location, strength, and quality of all light sources, along with the surface characteristics of each object in the scene including transparency, reflectivity, refraction, and shadows. Shiny objects reflect other objects, and realistic shadows are cast. Raytracing bounces the light through the model rather than terminating light values when they first intersect an object. The color and direction of light is changed by the object’s reflective and spectral properties.

## Reflection

A surface characteristic used to determine the extent which a material reflects other items in a scene.

## Reflection mapping

A method of simulating a complex lighting environment in which you treat a surface as a reflector and follow one ray (from your eye and reflecting off the surface) to select a point on a texture image (usually a rendering of the object’s environment) that defines the visual environment. As an object rotates in the environment, the image appears to move over the surface, in contrast to perhaps better-known texture mapping techniques, which fix an image on a surface. The resulting mirror effect adds realism and takes much less time to calculate than raytracing.

## Refractive index

The degree to which light bends (refracts) when passing through a transparent or translucent object. For example, the refractive index of air is 1.0 and water is 1.33.

## Scan-line rendering

A rendering method used to determine primary visible surfaces. The image is rendered one vertical scan line at a time rather than object-by-object as in raytracing. Scan-line rendering is faster than raytracing but does not produce as realistic results. LightWave 3D is a combination of both technologies; scan-line and raytracing. See also Raytracing

## Shader

See: Shading Model.

## Shading

Sometimes referred to as surfacing, this is the process of assigning values to the surfaces of objects. These values generally control how the surface interacts with light in the scene to create the object’s color, specularity (highlights), reflective qualities, transparency, and, if the surface is at all transparent, refraction. Basically, shading controls those qualities that suggest the material that an object is made of, whether wood, plastic, or metal. Most sophisticated shading techniques provide smooth transitions on curved object surfaces.

## Shading model

A “Shader” or Shading model is an algorithm used to define how an object’s surface reacts to light in terms of shading. Shading is the difference in color across a surface due to different surface properties and lighting. You can set shading according to such shading models as: Blinn, Cook-Torrance, Lambert, Phong, and etc.

## Specular

The highlights or simulated light-reflection created by light rays reflecting off a shiny surface. It is an important component of a material’s definition because it assists in revealing surface curvature or bumpyness in 3D space. Specular reflection depends on the position of the camera, whereas diffuse properties do not.

## Surface shader

Surface shaders come in different shading models (Blinn, Phong, Lambert, and Cook-Torrance) and define the basic color, reflectivity, and transparency of an object surface.

Also see: Shading Model.

## Texture mapping

The way a picture file is mapped onto an object to be used as a 2D texture. When a picture is mapped onto an object, the correspondence between the picture’s pixels and points on the object’s surface is calculated. You can map a picture file to the object’s XY, YX, or YZ coordinates. You can also map cubically, cylindrically, spherically, or by an object’s UV coordinates.

## Transparency

The amount of light that travels through a surface. Complete transparency allows all light through; no transparency makes the surface completely opaque.

## Transparency map

A texture map that varies the transparency and transparent color across a surface.

## UV coordinates

Two-dimensional coordinates that describe the position of any point on surfaces in terms of direction (U and V). UV coordinates are useful for mapping two-dimensional textures onto objects.

## Volume shader

A type of shader that controls what happens to rays that travel through a volume such as LightWave’s volumetric lights or Hypervoxels.



## Connection Table

	To Color	To Integer	To Scalar	To Vector	To Function
From Color	Color in RGB format unmodified.	CCIR 601 Luminance values as Integer Usually 0 or 1 only.	CCIR 601 Luminance values as float.	RGB values copied to vector components respectively.	N/A
From Integer	Integer value is duplicated to all 3 chans RGB.	Integer value unmodified.	Integer value unmodified.	Integer value copied to all 3 vector components.	N/A
From Scalar	Scalar values copied to all 3 chans RGB.	Rounded to the nearest Integer value.	Scalar values unmodified.	Scalar value copied to all 3 vector components.	N/A
From Vector	Vector components are copied to RGB respectively.	1st component ONLY is rounded to nearest Integer.	1st component ONLY is used.	Vector components unmodified.	N/A
From Function	N/A	N/A	N/A	N/A	Receive, transform, return.



---

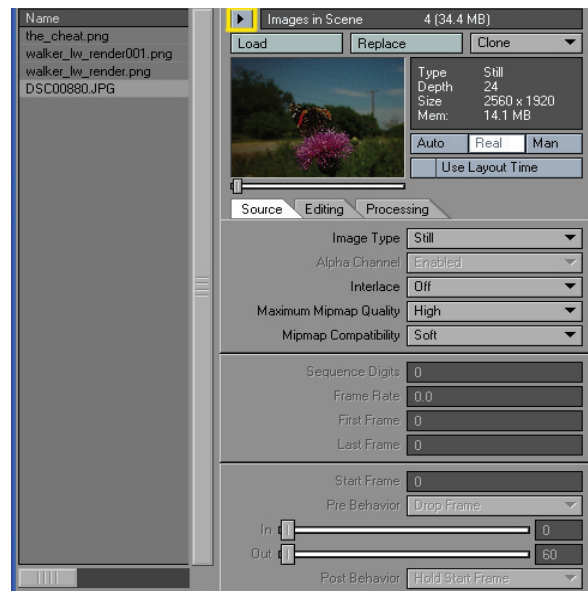
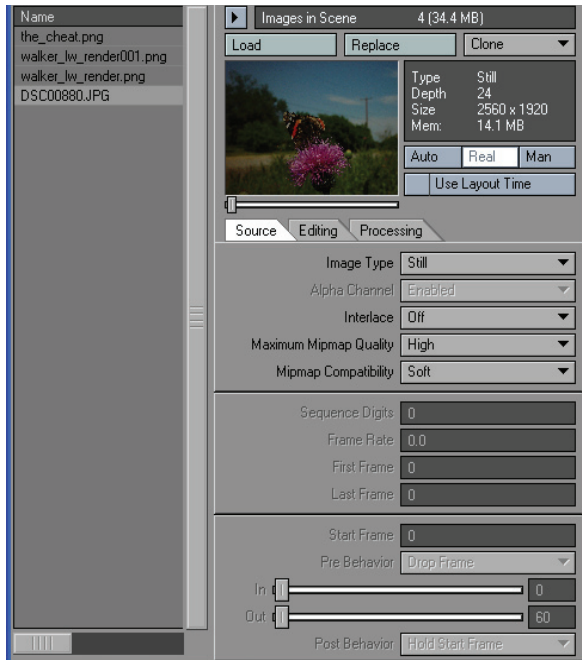
## Chapter 26: Image Editor

---

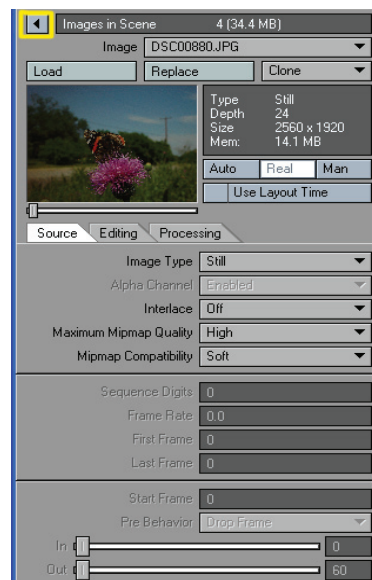


## Image Editor

LightWave's **Image Editor** is far more than a simple location to load up images or sequences of images to apply to your models. Of course, you can use it for that, but you can also edit the attributes for those images directly in the **Image Editor** without altering the original file, and you can set image sequence start and end frames within it.

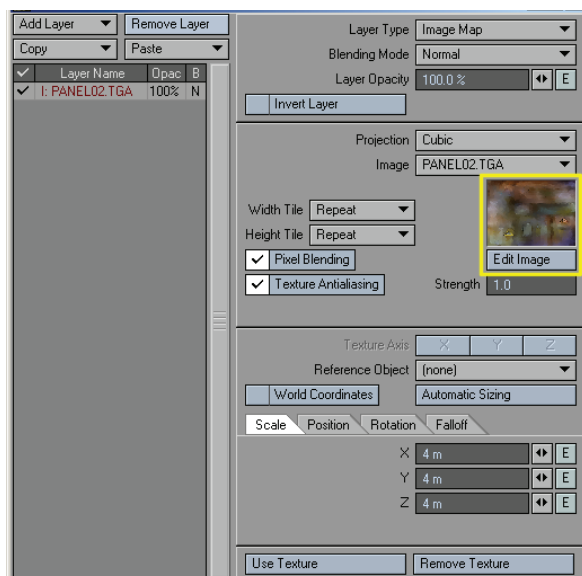


You can Collapse and Expand the Image Editor so that it takes up less screen real estate when you don't need it by clicking on the arrow that we've highlighted.



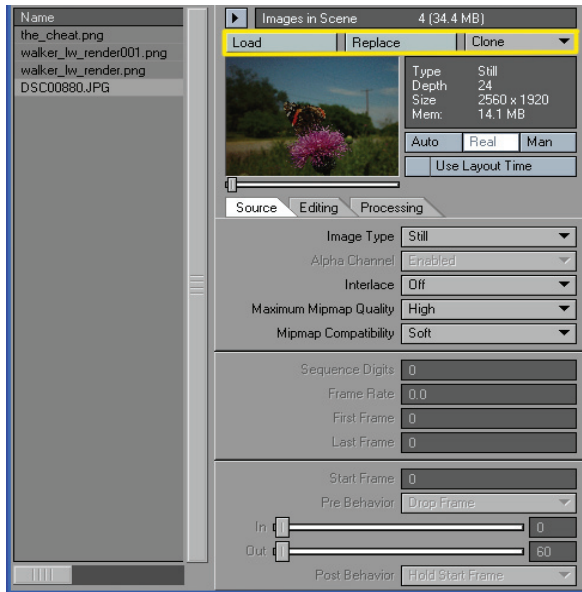
## Getting to Image Editor

Although you can hit **F6** to call up the **Image Editor** any time you like, sometimes it's nice to call it up with an image you are currently using, so that you can tweak for your particular needs. Any time that a window needs an image — particularly the **Texture Editor** — you will find an **Edit Image** button beneath the thumbnail of the image you've chosen.





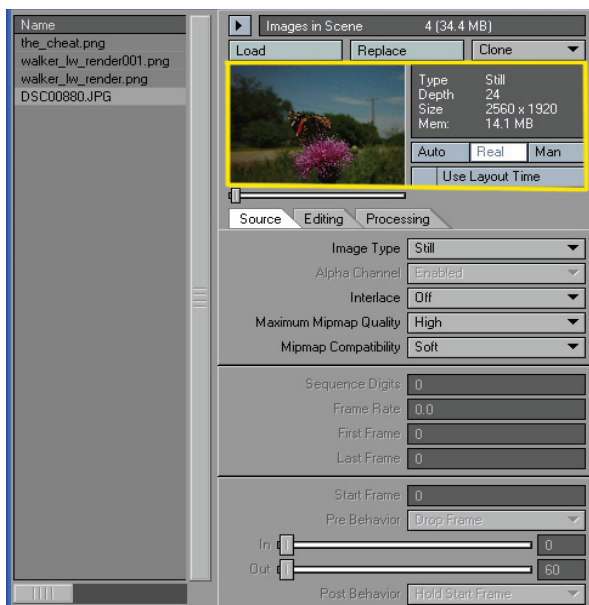
## Using the Image Editor



To load a new image or animation, or image sequence, press the **Load** button. To replace an image you've loaded, perhaps in error, press the **Replace** button. The **Clone** button is a drop down, because it has two different abilities. The first — **Instance** — copies the image, but allows you to work on with different settings in the **Editing and Processing Tabs**, almost as though it were a new image. Any changes you make to the **Instance** will not be reflected in the original image, however changes you make to the **Original** image will be reflected in all instances. If you are using an animation or an image sequence, any instances will have the same in and out points.

The second — **Duplicate** — can only be used on image sequences and simply copies the images. Each is treated as a completely separate sequence and changes made to the original sequence will not affect a duplicated sequence.

To delete an image, image sequence or animation, simply select it in the list on the left-hand side of the window and hit the **Del** key on the keyboard. You can also do it by clicking the right mouse button over the image in the list that you wish to remove.



The image preview window acts as a reference for you to see the effects you are having on your pictures when using the editing or processing functions. All edits are carried out on this 24-bit preview so that they can be done as close to real-time as possible, rather than working on the full image, scaling it down to the preview size and redisplaying it. Double clicking on the preview image will open the **Image Viewer** containing the image or frame with all processing and editing done. A large image and a great number of operations to perform on it may cause a delay in opening the **Image Viewer**.

The three buttons under the **Information Panel** give you options on how frequently you wish the preview image to update when a change is made. **Auto** updates the image when any change is finished and the mouse button released. **Real** updates the preview image in real-time, while you are making changes and **Man** means that any image changes are only performed manually, by clicking the left mouse button on the preview image.

The **Use Layout Time** button and the slider underneath the preview image are intimately linked. If you are using an image sequence or an animation you can use this slider to scrub through it. Alternatively, click the **Use Layout Time** button and you can use Layout's own timeline to scrub through your animation.

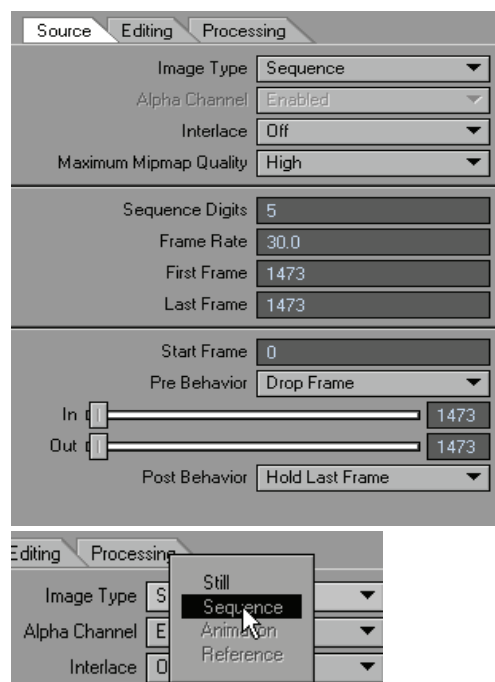
MIP maps are now only used to soften maps. You can control the exact amount of softening by editing the MIP Strength for the map: 1.0 is fully sharp, 2.0 is 2x filtering, 3.0 is 3x, and so on. This change presumes that all texture map anti-aliasing will be performed with pixel sampling. One major benefit of this change is that maps can now be softened when seen by reflection and refraction rays.

A toggle for using Classic mipmap handling or the new "Soft" mipmap option has been added to the Image Editor. For new scenes the default is Soft. Loading any scene that does not have Soft specified in the scene will use the Classic mode.



## Source Tab

The **Image Type** pop-up menu displays the type for the currently selected image. Generally, this setting is auto-detected when you load the file. However, you can specify a single image in a sequence of images by leaving its **Image Type** to **Still** and when you load a frame in a sequence that you want to use, don't forget to change the **Image Type** from **Still** to **Sequence**. Animation is used if you load an animation into the **Image Editor** and **Reference** is a clone of an image, as discussed above.



To be able to load a single image and tell LightWave that it is a sequence, you need to have a series of numbered files with the same filename. For instance, this is a valid sequence:

Ayslenn0001.flx, Ayslenn0002.flx, Ayslenn0003.flx and so on.

A sequence can be missing some individual frames, like so:

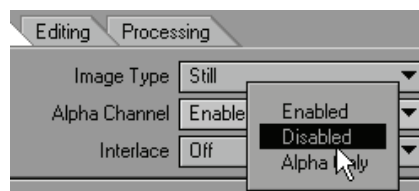
Ayslenn0001.flx, Ayslenn0002.flx, Ayslenn0007.flx,  
Ayslenn0008.flx

This sequence has frames 0003-0006 missing. When this is the case, LightWave will use the last frame it can to fill the gap. In other words, the sequence will be like so:

Ayslenn0001.flx, Ayslenn0002.flx, Ayslenn0002.flx,  
Ayslenn0002.flx, Ayslenn0002.flx, Ayslenn0002.flx,  
Ayslenn0007.flx, Ayslenn0008.flx

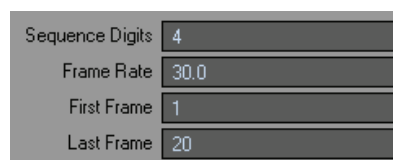
## Alpha Channel

If you are using images with embedded **Alpha Channels**, like some TIFFs, TGAs and Photoshop format images, you may wish to disable the **Alpha Channel**, otherwise, you may get unwanted effects since LightWave's **OpenGL** doesn't take embedded **Alpha Channels** into account. If you only wish to use the Alpha element of a picture, for a **Transparency Map** for instance, use the **Alpha Only** setting.



## Interlace

If you are using video grabs as still images, or a sequence, you may wish to use a field interlace. You have two choices, **Even First** or **Odd First** and which one you choose depends on your source material, but you will certainly know when you render if you've chosen the wrong one. If you are using images from almost any other source, you can leave **Interlace** to **Off**.



## Sequence Digits

When you load a sequence of images, LightWave automatically tries to ascertain the way the sequence is numbered by comparing files. You can override the number of digits used for a sequence if LightWave is getting confused.

## Frame Rate

**Frame Rate** controls the incrementing of an image sequence with respect to the scene's **Frame Per Second** setting on Layout's **General Options Tab** of the **Preferences Panel**. So, if you set this to, say, 15 and your scene **Frame Per Second** is 30, the image sequence will increment every two frames.





## Image Sequence Settings

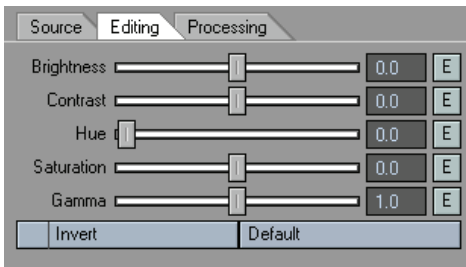
The **First Frame** and **Last Frame** settings display which files (or frames for an animation file) are treated as an image sequence in LightWave. This is computed when you load an animation file or switch the **Image Type** pop-up menu to **Sequence** (after loading a single image from the sequence), but can be set manually. However, it's best not to use these fields to trim your animation to fit your needs. Instead use the **In** and **Out** points discussed next.

The **Start Frame** entry determines on which frame in Layout your sequence will start. Use the **In** and **Out** sliders to determine how much of your animation you want to play back in Layout. The **Pre Behavior** and **Post Behavior** drop down menus determine what Layout should do when an animation is not as long as the frame count in your scene.



## Editing Tab

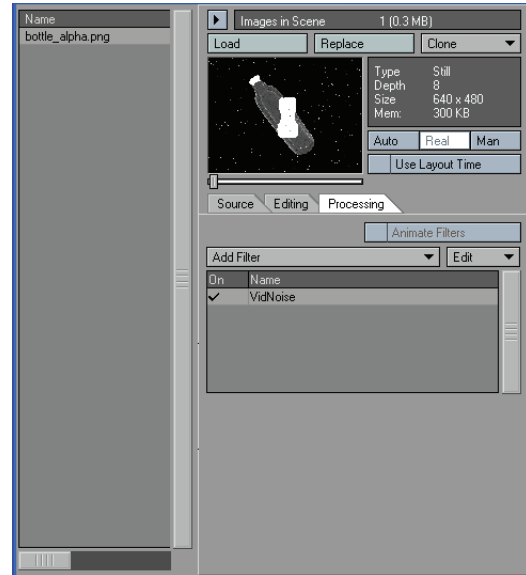
You can independently adjust (and envelope) various image parameters for the selected image on the **Editing Tab** if you want the images altered before LightWave uses them.



**NOTE:** These image operations do not affect the actual file on your hard drive, only the image used inside LightWave itself. If you wish to save your changes to an image, double click on the preview to open the **Image Viewer** and save the transformed image there in the format of your choice.

## Processing Tab

Access this tab to add image-processing filters. You may use any of the non-post-processing filters in the filter list. Ones that can only be used in the rendering process will inform you of the fact. Image filters added here will not update the preview window interactively unless the **Animate Filters** option is activated.







## Chapter 27: Rendering and Compositing



## Rendering and Compositing

A render is the image created by what the camera sees. A composite is a blending of two or more image layers. This chapter covers the details of creating the final image, sequence of images, video, and, any post-rendering effects you might add to the image

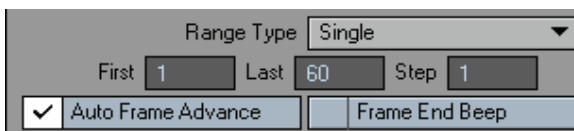
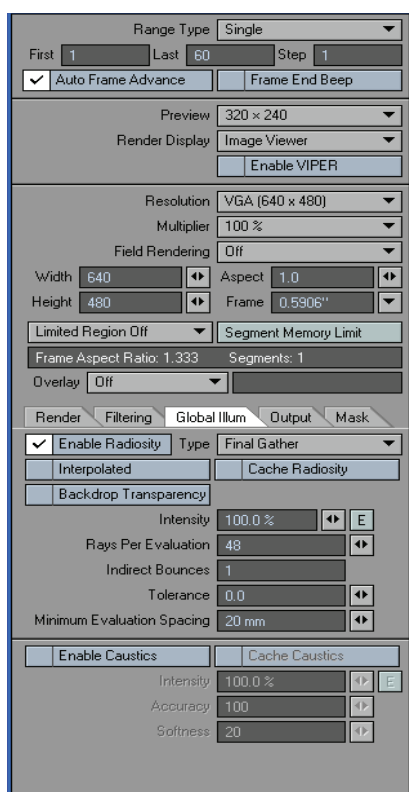
## Render Tab

## Render Globals

This is the heart of how you render your creation. In this window there are buttons, drop down menus and tabs to control how long your render will take, what format your individual frames or animation will be in.

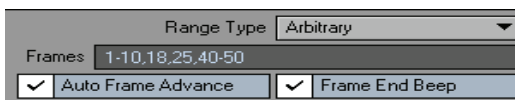
## Render Frames

LightWave 3D has three different ways of choosing which frames you wish to render when you press **F10**. You can choose **Single** — this allows you to set a range of frames to render the way existing users of LightWave will be familiar with. You set a first frame to render, a last frame to render and you can set a Frame Step. If you decide that you only want to render every other frame,



Arbitrary parses the frames you choose to render. You can set a frame range, a disparate single frame or two and a range again. To mark non-contiguous frames, put the frame number followed by a comma. To mark a range put the first frame number, a dash and then the last frame number in your range. Here is an example:

If you wish to render frames one to ten of your sequence, jump to frame 18, then render frame 25 and finish up by rendering frames 40 to 50 of your scene. This is what you would put in the Frames field:



The Keyframe sequence is very useful for creating storyboards. You have access to any object, light or camera in your scene and you can use the channels for movement, rotation or sizing for the item in question (you can't choose channels that don't exist — for instance, there are no size options for the camera). Every time that a keyframe for the item and channel appears in the scene, LightWave will render the frame.

The render range you choose is set independently from the scene frame range. If you use the Single range — the range that's always been available in LightWave — you will be asked to confirm if the range you've selected does not match the number of frames in the scene.

## Auto Frame Advance

By default, LightWave will only render one frame at a time, even if you have set a range or sequence. To carry on with your render sequence you can press the **Continue** button in the render progress display. This is not ideal if you are rendering a large number of frames, so there is a button called **Auto Frame Advance**, just below the Range Type in the Render Globals window that needs to be ticked for your render sequence to progress automatically. The Auto Frame Advance button is checked "on" by default.



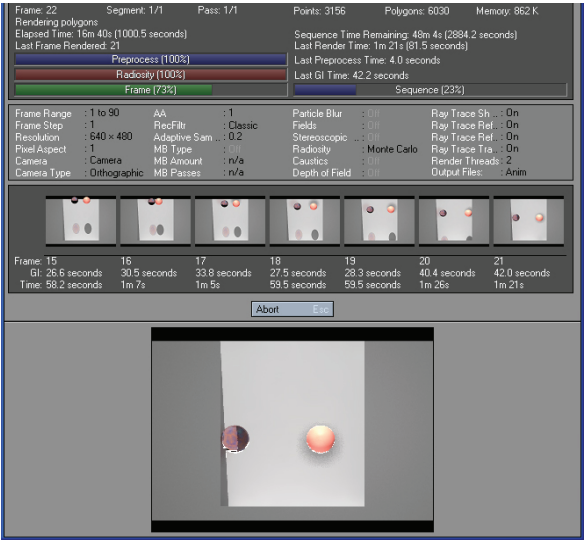
**WARNING:** If you are doing something else while rendering, make sure that auto frame advance is ticked before leaving your computer.

## Render Complete Notification

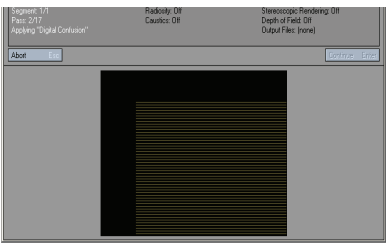
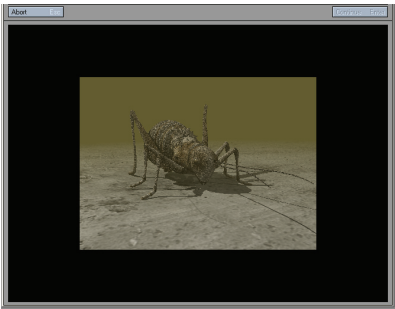
By selecting Frame End Beep, you can instruct LightWave to have your computer beep whenever a frame has been rendered or a preview has been generated. The status of Frame End Beep is saved as a default when you quit LightWave.



# Monitoring Progress

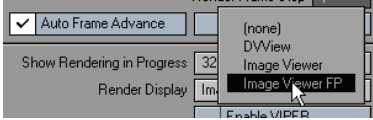


The frame progress is based upon the formula “(number of segments) \* (number of anti-aliasing passes)”. This means, for example, that if you have a render that is 1 segment with 5 anti-aliasing passes, you will see (1 \* 5 = 5) ticks on the frame progress indicator (i.e., 20%, 40%, 60%, 80%, 100%); if you have 3 segments with 9 anti-aliasing passes, you’ll see (3 \* 9 = 27) ticks on the frame progress indicator; and so on. If your frame has 1 segment and 1 anti-aliasing pass, then you’ll only see 1 tick on the frame progress indicator (when it goes from 0% to 100%).



The **Render Status Panel** that appears when you render can have a window added at two different resolutions to give you some indication of what is going on. However, if you are rendering an image that is bigger than the maximum size of 640 x 480, LightWave will not scale the render down to fit the preview window, for the sake of saving rendering time. This means that some parts of your image being rendered outside the window won’t be visible.

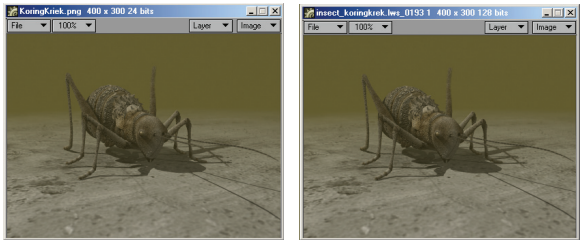
At any time you can cancel a render by hitting the **Abort** button or the **Escape** key. It may take a while for LightWave to respond depending on the speed of your machine and the complexity of the scene being rendered.



Before you start rendering, make sure you have the viewing option you want for the end of the render enabled. For rendering a sequence of images using **F10**, it’s certain that you will already have seen all your images and thus you can happily switch the Render Display to (none). Otherwise, you will want to see what you’ve done. Be warned that if you have no render display set, you won’t be able to change that once you start rendering. Also, know the difference between the Image Viewer and Image Viewer FP as discussed here and choose the one you wish to use before you start rendering. You will not be able to change Viewing choices during the render or after it has been completed without requiring a new render.

## The difference between Image Viewer and Image Viewer FP

LightWave can render your image to a precision far greater than your monitor will show. This is useful for print or film work. If you do a lot of post-processing on your images, having a full precision render is vital to preserve color gradients. LightWave has two image viewing tools: **Image Viewer** and the **FP version**. The **FP version** can be used to show the whole 192-bit range that LightWave renders to. With careful use of the **Image Controls > Expose** tools you can change the amount of light and darkness in an image without merely brightening or darkening it, preserving and even revealing detail. **Image Viewer FP** does take more memory than the standard 32-bit **Image Viewer**, so in situations where memory usage is an issue you may want to use the 32-bit viewer. Be aware that although image formats which support floating point color are available in the standard **Image Viewer** they will only be saved as the 32-bit images that the standard **Image Viewer** displays.



## DV View

Before we move onto the next section, we need to look at another viewing option on the drop down: DV View. Choosing this allows you to display your renders directly on a DV device attached through a Firewire port on your computer. You can output to a DV video camera that is connected to a video monitor to allow you to see your work directly on a TV screen. This viewing option only works on NTSC systems, not PAL ones.

## Render2Image

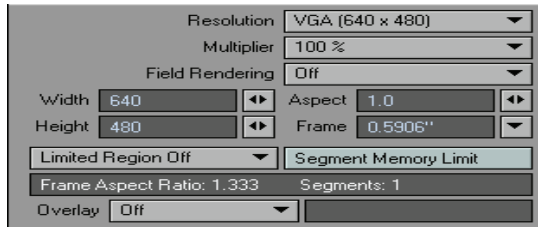
Part of Visor, this option allows you to render images directly to the Visor pane. See the Visor section later in the chapter for more information.



## Enable Viper

In the **Render Tab** menu, you can enable **VIPER** by clicking on the so-named button (**Render> Enable VIPER**). Be sure to turn it off when you are ready for your final render since it takes memory and processor time, slowing down your renders.

## Resolution



The Resolution Preset drop down menu will present you with a series of pre-defined resolutions to choose from for your render. It will automatically set the Width, Height and Pixel Aspect ratio fields.



**NOTE:** You can add your own presets to this list, but it requires you to delve into the LW9.cfg file. If this thought doesn't scare you, then we'll proceed.

If you look in your LW9.cfg file you see that near the top there are several lines that look like this:

```
ResolutionPreset 1920 1080 1 0 0 1920 1080 HDTV (1920 x 1080)
```

The first two numbers are the size of the frame, the next one is the pixel aspect ratio and then the next four are the default Limited Region frame. The last bit of text is the title of the preset that will appear in the drop down menu.

Feel free to make your own lines. For instance here's one you may wish to add to get the ball rolling:

```
ResolutionPreset 2480 3508 1 0 0 2480 3508 A4 page (300dpi)
```

As you can tell from the title, this Resolution Preset gives you a full A4 page at 300dpi.

The width and height fields can be set to anything between 16 and 16,000 pixels. Be aware that larger resolutions can make serious demands on the memory of your machine.

The Resolution Multiplier gives you a much more consistent way of quickly checking a scene rather than changing the width and height fields when you want a small test render. It takes into account the scaling of things such as particle, line, and edge thickness, as well as the glow radius.

If you have selected a resolution preset and you alter the width or height fields, it will override any preset and the menu will then show the word Custom. If you have already set a resolution multiplier, it will then operate on the Width and Height settings you have chosen.

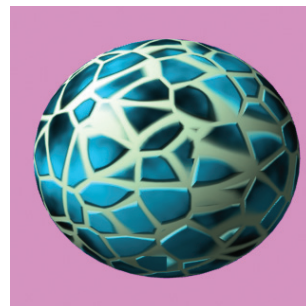
## Pixel Aspect Ratio

Once you've been using a computer for a while you forget that pixels

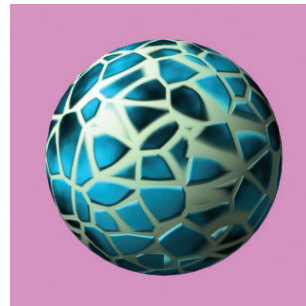
actually come in different shapes. Ones for NTSC TV are tall and thin; ones for PAL TV tend to be a bit fatter, while ones for print are the same as those for computer screens – square as square can be.

The Pixel Aspect Ratio setting in LightWave is calculated by dividing the width of a pixel by its height. A pixel intended for print or a computer screen is square, as we said, so its aspect ratio is 1.0. Because NTSC pixels are taller than they are wide, the aspect ratio tends to be between 0.86 and 0.9. PAL ones, on the other hand, tend to vary between 1.01 and 1.06. Values for widescreen displays are considerably wider in both NTSC and PAL.

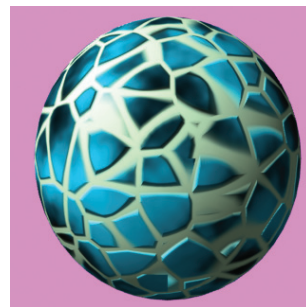
Why worry about the pixel aspect ratio? After all a pixel is a pixel, right? Well yes, but if you look at a perfectly round ball that has a radius of 50cm and you are using an NTSC resolution preset, the ball will look squashed on a computer monitor, whereas it will look perfectly round on your NTSC monitor. When selecting one of the resolution presets you will notice that the pixel aspect ratio changes along with the resolutions for width and height. As for things looking squashed or stretched on your computer monitor, I'm afraid it's either something you'll have to get used to, or you will need an output to a proper broadcast monitor to reassure yourself.



NTSC (0.9)



Computer monitor (1.0)



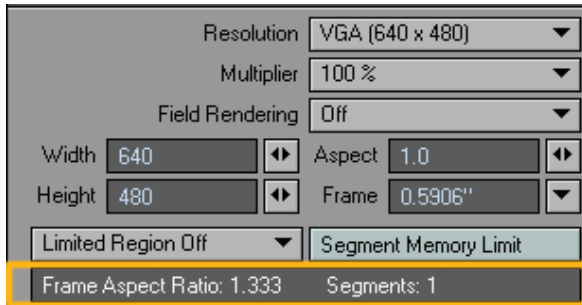
PAL (1.0667)

Same ball, different monitors.





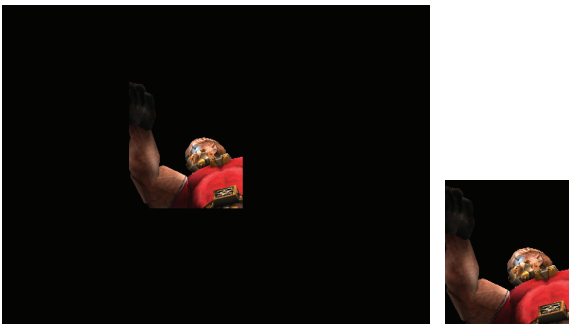
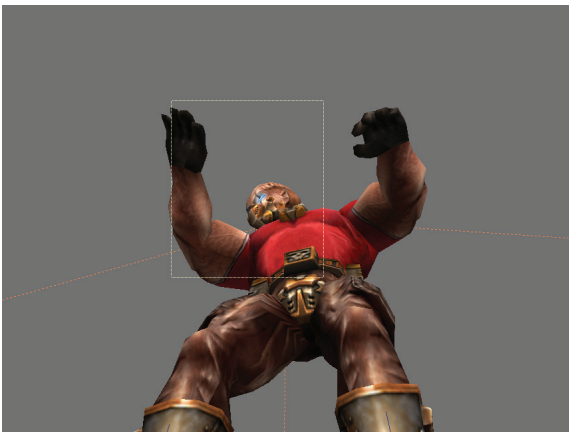
## Frame Aspect Ratio



Before we move on, don't confuse the pixel aspect ratio with the frame aspect ratio figure, often referred to simply as the aspect ratio. The way to work this out is to take the pixel width of a picture, divide it by the pixel height and multiply the result by the pixel aspect ratio. As an example, a standard VGA screen is 640 x 480. This equates to a frame aspect ratio of 1.333, which is the result of the following sum  $(640/480) \times 1$  and converting it to a ratio. You will often see this figure quoted on the back of DVD cases to indicate the width of the display compared to its height (which indicates how much of your TV screen will be covered by black bars).

### Rendering a limited region

You can render a limited rectangular region of your scene if you only wish to test your work, rather than taking the time to render a full image. Simply hit the L key and you will see that a dotted line will surround the renderable area of the frame. You can change the size and shape of this area by clicking and dragging the left mouse button on one of the dotted lines that surround the area.



Above: What Layout looks like, Middle: Render Limited Region Borders, Right: Render Limited Region No Borders

There are two different types of limited region that you can use, either with or without borders and you cycle through these choices by repeatedly hitting the L key or by choosing the drop down menu

in the Camera Properties window. The difference between a limited region with a border and one without it is the fact that a limited region with a border puts your limited region on a black page the size of a full render, whereas a limited region without borders will just render the shape you desire as the full image. The frame aspect ratio in Camera Properties will remain at the aspect ratio for a full frame, but all other options, such as antialiasing and Masks still apply.

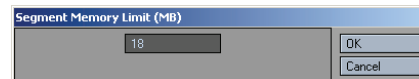
The limited region with border allows you to "patch" only a segment of a frame, rather than having to re-render the whole frame for re-compositing. This can be a major time-saver.

### Memory Considerations

Limited Region allocates only enough memory to render the horizontal limited region area. If you stitch parts of an image together, you can effectively render images that are much larger than those you could render one pass. This is especially useful for high-resolution print images or in low memory situations. However, note that some post-processing filters require a full-sized image. In such cases, you may be able to apply those filters to the "stitched" image in an additional step. The way to do this is to take your final rendered image and save it to disk. Then clear your scene – or better yet, quit and restart LightWave – and load this image into an empty scene. Make it the camera backdrop and add whichever post-process filter you wish to use, and then render again. Since you aren't rendering all the objects, textures, Image Maps, etc., the memory requirements will be a lot lower.

## Segment memory limit

When rendering an image you can dictate the amount of memory that LightWave should devote to rendering — this is apart from whatever memory LightWave uses for Shadow Maps, image mip-mapping or geometry memory costs. The default value for Segment Memory Limit is 64 MB, but changing it will result in a question of whether the new value should be set as the default. A value of 18MB is enough to render a full frame at video resolution (either NTSC or PAL) in a single segment. Setting it higher than this will not result in LightWave devoting more memory to render the image, the limit is just that. If additional memory is assigned and not needed it will not be used. The Segment Memory Limit can be set as low as 1 MB, useful for rendering scenes with particularly stringent memory requirements.



There are other reasons for setting a memory limit to be enough to render a frame in a single segment. The first is that an image rendered in a single segment renders faster than one with multiple segments. The second is that some image filters require an image to be rendered in a single segment, otherwise unsightly borders can occur. For instance this is the case when blurring occurs right up against the edge of a part of an image that becomes visible when the image is assembled.



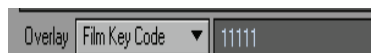
**NOTE:** On platforms that support virtual memory, you may get better results using smaller segments that fit within available RAM. (Using one segment that may not fit entirely in RAM forces you to page to the hard disk and slow down rendering). You may need to experiment with segment values to find a useful setting.



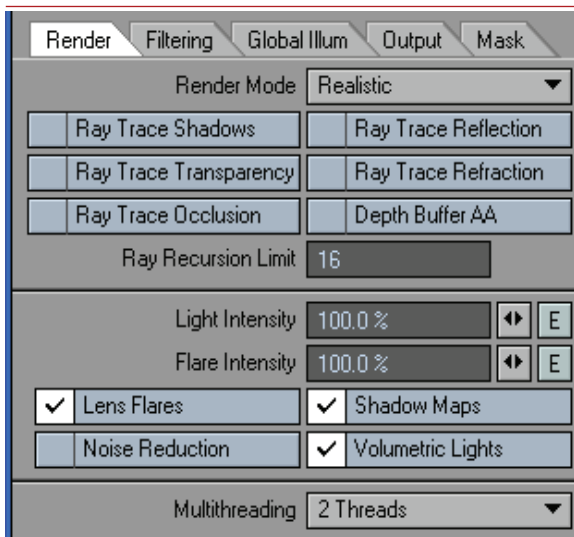
**HINT:** When you turn off Adaptive Sampling, images take longer to render, but you will achieve better antialiasing results. If using Adaptive Sampling does not give you the results you wish or fine lines are being missed (even at a low or zero sampling Threshold level), disable Adaptive Sampling.

## Data Overlay

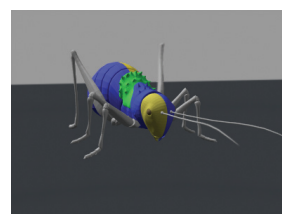
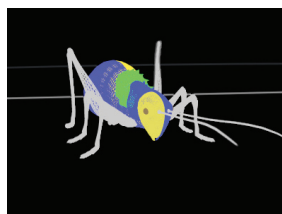
The Data Overlay option lets you place a descriptive title (of up to 20 characters) along with a frame reference in the lower portion of the rendered image. This is useful when you generate large numbers of animations for others to review and approve before final rendering. It can serve to identify the scene being rendered, and the specific frames you may wish to change. The Data **Overlay** pop-up menu setting determines the format for the frame reference. Enter the text in the **Label** field. If **Data Overlay** is not set to **Off** and the **Label** field is blank, the scene name will be used automatically.



## Render Globals: Render Tab



Although you may usually want to use Realistic mode to calculate your renders, don't forget **Quickshade** and **Wireframe**. Quickshade only uses the basic colors for your objects and ignores any **Image Maps** or other types of surface properties (including smoothing, textures, transparency settings or object dissolve levels). Wireframe goes one step further and doesn't even show you filled polygons! Both options respect the Subpatch levels in Object Properties. Therefore, if you want a very smooth surface, you can increase the Subpatch level and have it reflected in your render. Make sure you do not increase subpatch levels too much when rendering in Wireframe mode. The number of polygons may well increase so much that you end up rendering a silhouette! Wireframe mode uses the basic color properties of a surface to determine what color it should render the wireframes using that surface.

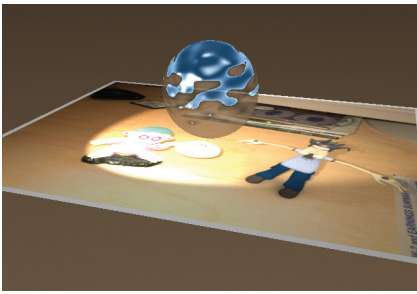


**Quickshade** is very useful for blocking out animation where the detail of an object may become distracting, or for CPU-intensive features like Motion Blur and Depth of Field where its speed will become apparent.



**HINT:** To produce a solid wireframe animation, Use the Unshared Edges, Surface Borders, and Other Edges options on the Object Properties Panel's Edges Tab, and render in Quickshade Mode.

The last option **Realistic** will be the one you use most often. You can use various ray tracing options, but by default they are turned off since they will each add to your render time.

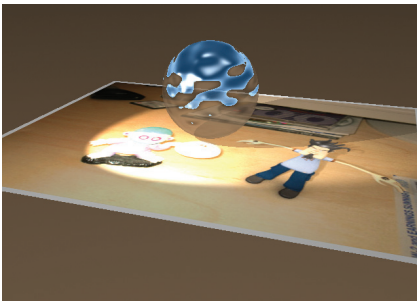


Ray Trace Shadows

When you enable “Ray Trace Shadows”, any lights that have ray-traced shadows enabled will now calculate ray-traced shadows.

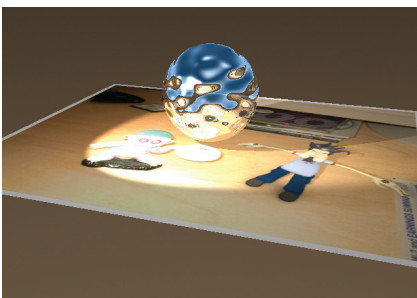


**HINT:** To optimise your rendering times, you should carefully consider which shadow options you wish to have active for each object – for instance, objects that are in the background can probably have their shadows turned off completely.



Ray Trace Reflection

Enabling this feature will cause LightWave to calculate ray trace reflections between objects and the background depending on the settings for the surfaces in the **Environment Tab** in the **Surface Editor**. Again, you can optimise your rendering times by using images as **Reflection Maps** instead of raytraced reflections or by turning off reflections altogether for objects with lower importance.

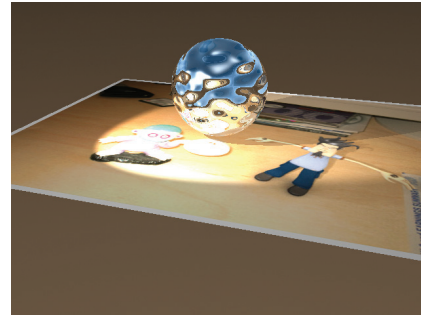


Ray Trace Transparency

Enabling this feature causes all transparent surfaces in your scene to be ray traced, not just ones with an index of refraction different to 1.0. This allows volumetric effects like **HyperVoxels** to be seen behind transparent surfaces without having to make the surface refractive as well.

## Ray Trace Refraction

You can get LightWave to ray trace only transparent surfaces with refraction using this feature, allowing you to further optimise only those surfaces that have to undergo the lengthy ray tracing procedure. It works for surfaces that have some degree of transparency and a refractive index other than 1.0.



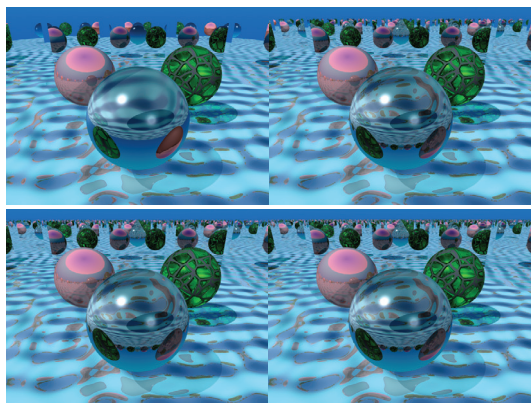
**NOTE:** Selecting Unseen by Rays for an object tells LightWave to ignore the object in its ray-tracing calculations when reflection and refraction are involved. This means that the object that is unseen by rays will not appear in the reflections or refraction of another object. It will, however, render normally in the scene. This is especially handy for objects that are front projection-mapped; you probably do not wish them to show up in the reflections within other objects. Unseen by Rays will not affect the shadow options of a given object.

Allows you to enable occlusion when the ray recursion level is reached. If the ray is occluded, the background will not be seen by the ray. This should work for all reflection, refraction, transparency and dissolve rays regardless of their origin (LightWave, plug-ins or Node Editor). Rays can be partially occluded when they pass through partially transparent or dissolved surfaces. Occlusion also considers clip maps. Ray Trace Occlusion will save users from having to create reflection masks to remove the background from areas where it should be blocked by other objects.



## Ray Recursion Limit

Ray Recursion is the maximum number of times you want ray-traced reflections reflected off reflective surfaces. In the real world, things can be reflected a seemingly infinite number of times. For instance, if you stand between two mirrors facing each other you will see a seemingly countless number of reflections, repeating until the imperfect transmission of light due to the density of the glass and air causes them to fade into blackness. In the world of 3D animation, you must set a limit for the number of reflections. The default value is 16, but you can set this field from 0 to 24. Lower numbers will yield faster rendering times.



Ray Recursion set to 3 - 2:12, Ray Recursion set to 8 - 3:42, Ray Recursion set to 16 - 4:26, Ray Recursion set to 24 - 4:51.



**NOTE:** As a default, try to use a lower rate of ray recursion than the standard 16, it will make your renders a little bit faster. Going below 3 can cause problems with things such as Clip Maps, so it's always a good idea to have some Ray Recursion in the Render Globals.

## Shading Noise Reduction

The **Shading Noise Reduction** option is designed to reduce the graininess in the shading of diffuse surfaces from linear/area lights or using radiosity with low **Tolerance** values. This option will add some time to rendering, but will result in smoother shading. For radiosity, the alternative of using a higher **Rays per Evaluation** setting would add much more to rendering time.

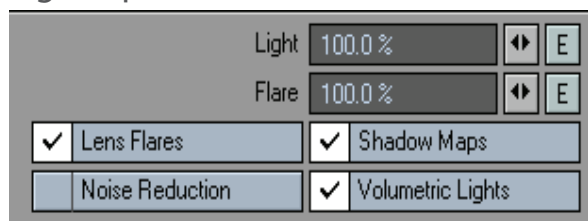
## Multithreading

The **Multithreading** setting is used for systems with multiple CPUs. Normally, you should set this to the number of processors on your system; however, there are times when you will get faster rendering with this set at a greater or lower number. For scenes that are very simple and render in seconds, you may find that setting multiple processors to the task will actually slow down renders because of the overhead of the communication between the processors. For such scenes, it is better to set up batch rendering so that each processor can render its own frames. In the case of long frame renders, you will sometimes find that setting the multithreading to a number greater than the number of real or virtual processors in your computer can help. In either case, you may want to test a few frames before beginning your final render.



**WARNING:** A potential for problems exists with some plugins that were not designed with multithreading in mind. If problems occur, try using the 1 Thread setting.

## Light Options



**Light:** Determines the Global strength of all lights in the scene. A setting of 100 uses the same setting, a setting of 50 would cut the strength in half, while a setting of 200 would increase the strength two-fold.

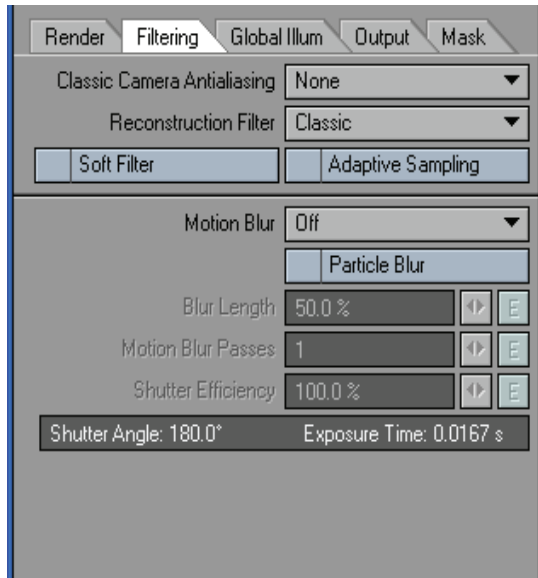
**Flare:** Determines the Global strength of all Lens Flares in the scene. A setting of 100 uses the same setting, a setting of 50 would cut the strength in half, while a setting of 200 would increase the strength two-fold.

**Lens Flares, Shadow Maps, Noise Reduction, Volumetric Lights:** Checked, these items will activate all active items of that type, while unchecked will turn off all items of that type.





## Render Globals: Filtering Tab



See Chapter 13 for more information on the antialiasing settings.

## Motion Blur Effects

When using a camera to film fast-moving objects, these objects are often blurry. This is because they continue moving while the shutter of the camera is open. It is this feature that LightWave aims to replicate with Motion Blur.



Motion blur becomes essential when animating, especially for use with live action. It prevents the crisp quality that normally pervades computer-generated animation and helps an animation appear more fluid.



Use a camera view and render a Motion Blur preview by hitting Shift F9.

LightWave's motion blur system takes everything that can change over time into account. From shadows, to surfaces, from light intensities to object or camera movement. It accounts for curved motion and does not blur in a linear fashion, but rather following the path that the motion is taking.

For motion blur to work, some level of antialiasing needs to be enabled. LightWave uses these antialiasing passes to generate the additional images used by motion blur. You will be able to see the process working if you are rendering in a render view. For each antialiasing pass, LightWave seems to move the objects a little and then composites them all together to get the motion blurred image. Because only five steps (a low level of antialiasing) can give a stepped effect, higher levels of antialiasing are recommended. There are three types of motion blur — normal, dithered and Photoreal. Dithered provides a better quality result with double the number of images to dither in between, and doesn't take as long as using the next level of antialiasing, but provides results just as good if not better. Introduced in LightWave v9.2, Photoreal provides the best quality blur. Instead of rendering a new pass for each motion blur sample, Photoreal Motion Blur allows for multiple samples within each render pass.



**NOTE:** Photoreal Blur will not work with the Classic Camera. You must use one of the Advanced Cameras, such as the Perspective Camera



**HINT:** Using Soft Filter in combination with Dithered Motion Blur creates an even better effect.

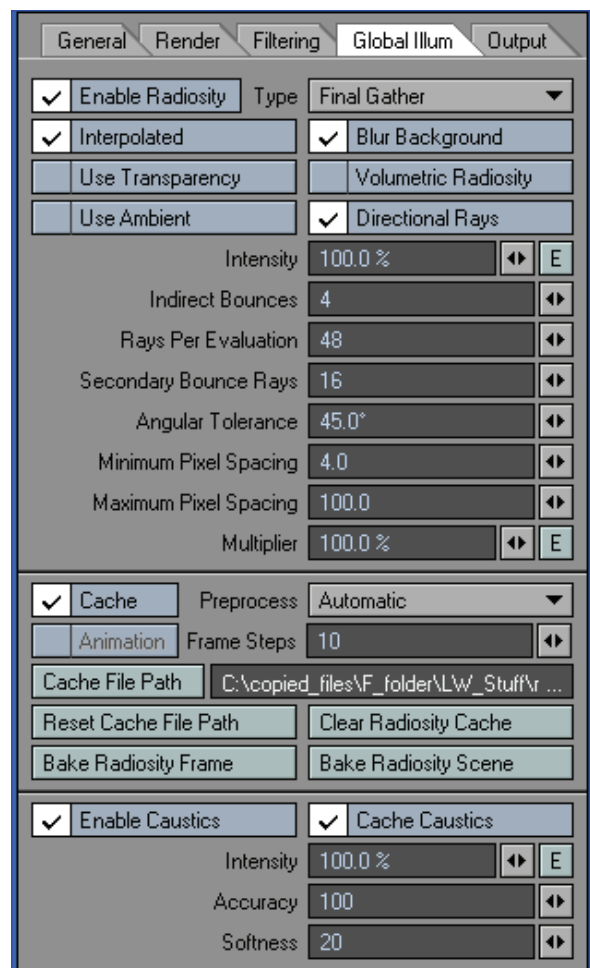


**NOTE:** For more information on Motion Blur, see Chapter 12: Camera Basics.



## Global Illumination Tab

You access the **Global Illumination Tab** by choosing the **Global Illumination** tab on the **Render Globals Panel**. Here you can adjust some of the global lighting controls, including radiosity and caustics.

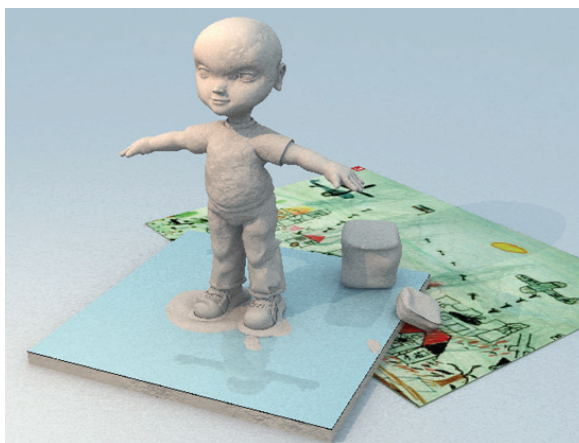


The **Global Light Intensity** value is an overriding *volume* control for all lights. 100% is the normal setting. However, you can ramp all of your lights up or down, or even create an envelope. The **Global Lens Flare Intensity** is a similar control for lens flares.

You must activate **Enable Lens Flares**, **Enable Volumetric Lights**, and **Enable Shadow Maps** when you are using any of these features. (You may find it quick to toggle these settings from the **Render Globals>Render Tab** menu.) These *switches* are quick ways to turn these options off globally for test purposes, without losing any of your settings. However, note that they do not turn off the light sources themselves.

## Radiosity

Without using LightWave's radiosity option, all surfaces are lit *directly*, with lights or ambient light. *Radiosity*, the scattering (reflection) of light off diffuse surfaces, causes surfaces to become (indirect) light sources — like in the real world — generally resulting in much more realistic images.



In LightWave, the scattered light includes surfaces that generate their own light through use of the **Luminosity** surface attribute. The images that result from a radiosity renderer are characterized by soft gradual shadows. Radiosity is typically used to render images of the interior of rooms, due to the high amount of bounced light, and can achieve extremely photo-realistic results for scenes that comprise diffuse reflecting surfaces.



Image by Andrew Bradbury - Copyright Andrew Bradbury

LightWave can calculate secondary rays bouncing from surfaces or coming from the atmosphere. This adds a very subtle, but photo-realistic effect. When combined with the *high dynamic range* calculations, the renderings become astoundingly realistic.

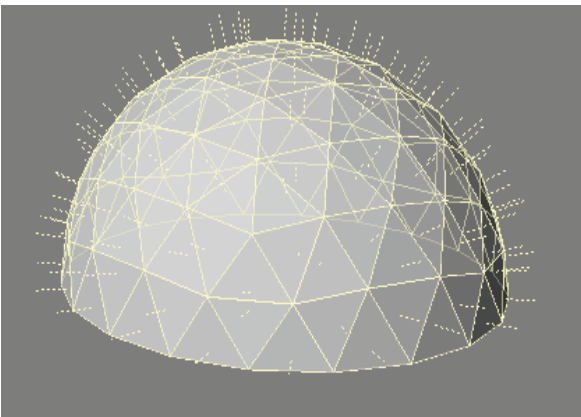




Copyright Philip Meyer

## How Radiosity Is Calculated

LightWave approximates radiosity using “projection hemispheres.” These basically “sit” on surfaces, each projecting out multiple radiosity rays at various angles using the theoretical normals of each polygon in the hemisphere. This is called an “irradiance evaluation.”



An Illustration of a Projection Hemisphere

If a radiosity ray strikes a surface that scatters light, some amount of that light illuminates the surface where the ray originated — colors are determined in the usual (non-radiosity) way, which can include the effects of luminosity, mirror reflections, caustics, and so on. The light-scattering surfaces are essentially extra little light sources, used instead of ambient lighting, that affect the diffuse shading of the current surface. Evaluated and non-evaluated areas are blended to compute the final effect.



NOTE: If you activate the Unseen by Rays option on the Rendering Tab of the Object Properties Panel, that object's luminous surface is not considered a source of light for radiosity purposes.

Performing a full irradiance evaluation (tracing hundreds of rays) every time a point must be shaded is too time-consuming. Fortunately, unlike direct lighting, with its concentrated sources that can cause sudden changes across a surface (e.g., shadow boundaries), the indirect lighting that radiosity is meant to handle tends to change gradually over a surface. LightWave can use the results of each previous evaluation and smoothly interpolate between them. Only

when no previous evaluations are close enough to the point being shaded, is it necessary to fire a new set of rays.

What is *close enough*? Each time LightWave performs a full evaluation, it estimates the size of the area that can produce valid results. This depends on factors like how far the rays travelled. If all rays went a great distance before hitting anything, then the indirect light must be fairly constant and the calculated irradiance should be good for a large area. But if several rays hit nearby objects, then the indirect light might be varying more rapidly across the surface and the irradiance evaluations should be more closely spaced.

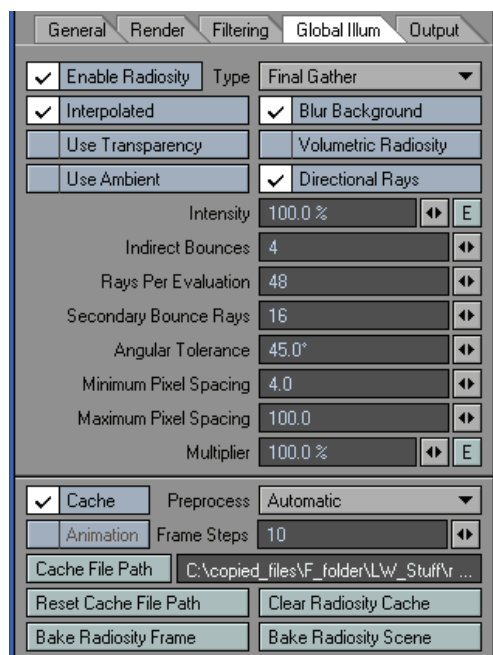
The **Tolerance** setting is a scale factor on these valid area estimates. With a large **Tolerance**, the renderer reuses previous irradiance values more often rather than computing new ones, and rendering time is reduced at the expense of accuracy (i.e., in some areas, small local changes in indirect lighting might be missed). With lower **Tolerance** values, full hemispherical evaluations are performed more frequently — a zero value won't interpolate at all (always do a complete new evaluation). As such, **Tolerance** acts as a limit on the amount of error allowed in the radiosity calculations, although there will always be some errors due to the use of a finite number of rays.

Several Global Illumination default values have been changed. This should make it easier to get started with radiosity.

- \* The mode will now default to Final Gather.
- \* Volumetric Radiosity is disabled.
- \* Directional Rays is disabled.
- \* Interpolation is enabled.
- \* Use Transparency is disabled.



## Radiosity Settings



Activate **Enable Radiosity** for LightWave to render this phenomenon.

In order of increasing complexity, the radiosity **Type** choices are: **Backdrop Only**, **Monte Carlo**, and **Final Gather**. **Interpolated** allows you to set the Tolerance value. **Backdrop Only** evaluates only the rays that hit the backdrop. It is faster than **Monte Carlo** for environmental illumination, especially in scenes with “expensive” textures or lighting. However, it does not account for diffuse inter-reflection, luminous surfaces, and so on. **Backdrop Only** mode is the same as an occlusion render such as you would get using gMIL on all textures.

**Final Gather** calculates the rays cast from the illuminated points on a surface, from which a hemisphere created with a specific radius and calculates the direct and indirect illumination. **Final Gather** stores shaded samples in world space but only for secondary rays. **Final Gather** only shades the surface if there are no other **Final Gather** samples near the point that was it (within the **Minimum Evaluation Spacing** and **Tolerance**). When the **Final Gather** does shade the point, it also adds it to the **Final Gather** samples so that if a point near it gets hit again, it won't need to be re-shaded. This is why **Final Gather** is faster than **Monte Carlo** but takes more memory. **Final Gather** can be slower than **Monte Carlo** or even run out of memory if the **Minimum Evaluation Spacing** is too small.



Note: If you have your Minimum Evaluation Spacing set too small, the preprocess of Interpolated can be slower than Final Gather. Try making the Minimum Evaluation Spacing larger. Also, if the tolerance with Interpolated is set too low the same thing can happen.

All radiosity types have had an **Interpolated** mode added as an option. Interpolated only works with the primary rays (surfaces seen directly from the camera). It can use either Monte Carlo or Final Gather rays to generate a radiosity illumination value at the point hit. It generates a hemisphere of these rays, which are evenly distributed but randomized to some extent.

The **Directional Rays** option defaults to on in existing and new scenes. When this option is enabled, the radiosity will include illumination from

directional sources such as reflections, refractions, transparency and fog. When the option is disabled, the only illumination will come from non-directional sources such as diffuse.

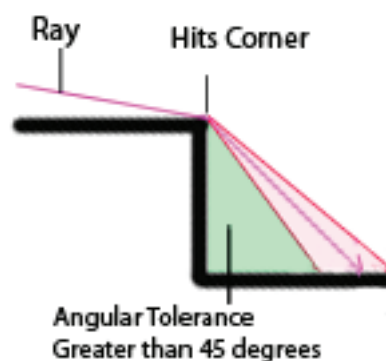
**Blur Background** provides you with a way to use GI settings that provide a quicker render, but still have a smooth background in the image.

**Cache Radiosity** saves radiosity data for subsequent render passes and frames, which can significantly reduce rendering time. The results can be inaccurate if objects or lights are animated, but this option works particularly well with scenes like a walk-through in which only the camera moves.

The purpose of **Use Transparency** is to disable transparency calculation, including clip maps, for backdrop radiosity rays. **Backdrop** radiosity is supposed to be a fast and simple form of global illumination. Having it do a lot of transparency calculations makes the image slightly higher quality but if you want higher quality you will probably be using one of the other modes anyway. This option is only available for **Backdrop** type radiosity. When **Use Transparency** is disabled for Final Gather, all transparent surfaces are ignored by rays. This acts the same as putting all the transparent surfaces into an object and setting it to be unseen by radiosity.

**Intensity** determines the overall amount of radiosity, with a default of 100%.

The numeric value for **Angular Tolerance** is set in degrees. Angular Tolerance is the amount +/- in degrees in which the rays are allowed to vary after hitting a surface. This should be easier for you to control and is more compatible with other rendering programs (such as K-Ray) that use degrees to control this value. The minimum angular tolerance is 10 degrees, to allow for better sample blending. When the Angular Tolerance is set to more than 45 degrees, the behind test will be disabled. For highly uneven surfaces (such as trees or asteroids) this will greatly cut down on the number of radiosity samples in the scene and improve rendering performance. The behind test is used to prevent samples on steps from blending with samples on other steps.



**Rays Per Evaluation** represents the number of sides and segments of the projection hemisphere (as when you model a ball), which determine the number of radiosity rays sent out for evaluation. As you might expect, the higher the density, the more accurate, but the longer rendering will take.

**Indirect Bounces** determines the number of times an indirect ray from a light source on a surface will be calculated. Up to 24 bounces are available.



Note: Ray Recursion is the upper limit on the number of Indirect Bounces you are allowed. For example, if you have Ray Recursion set to 1 and Indirect Bounces set to 8, only one radiosity bounce will be calculated. However, if you have Ray Recursion set to 8 and Indirect Bounces set to 4, you will still get 4 bounces of radiosity.

A setting of 0 in the **Minimum Pixel-Size (MPS)** control means that pixel size will not be factored into sampling.



NOTE: In internal testing, Minimum Pixel Spacing of 1 or less eliminates “light leaks” and produces sharper shadows

**Maximum Pixel Spacing** controls the maximum distance in pixels between interpolated radiosity samples. The value should be fairly large (100 pixels or more) to speed up rendering. Large values can also cut down on visible noise in large flat areas of the scene.

A Volumetric Radiosity checkbox has been added to the Global Illumination Tab in the Render Globals Panel. Volumetric Radiosity was previously added in the Edit Menu Layout Panel.

Also, a **Use Ambient** checkbox has been added. When this option is selected, the ambient light is added to the background color when computing radiosity. This allows the ambient light to be occluded. You can illuminate your scenes with just ambient light now if you wish. The ambient color will not affect the background color in your rendered images.

## Cache Radiosity

Saves radiosity data for subsequent render passes and frames, which can significantly reduce rendering time. The results can be inaccurate if objects or lights are animated, but this option works particularly well with scenes like a walk-through in which only the camera moves.

The Always Preprocess option will force the radiosity to be preprocessed even if the current frame already exists in the cache.

The radiosity cache now gets loaded before baking begins if it exists. This will allow you to continue baking an existing radiosity cache. You could, for example, extend an animation by setting your start and stop frames beyond those you have already baked and clicking Bake Radiosity Scene. You could add just a single frame to the cache by selecting a frame and clicking Bake Radiosity Frame. The downside to this change is that if you want LightWave to bake the radiosity from scratch, you must first click the Clear Radiosity Cache button before baking.

If your chosen Radiosity mode set has Interpolated checked, the Cache Radiosity option is available.

**Cache Radiosity:** Turns on the ability to cache radiosity.

**Disk Cache:** Saves the cache to disk at the specified directory.

**Cache Directory:** The cache file name is based on the scene name. If your scene is Test.lws, the cache file will be named Test.lws.cache and placed in the folder you specify. When you click the Clear Radiosity

Cache button, only the cache file associated with the open scene will get deleted now. This will allow you to use the same radiosity cache folder for multiple scenes.

**Reset Cache Directory:** Set the cached directory back to the default, if the user changed it to something else.

**Clear Radiosity Cache:** Deletes all files in cache directory.

**Bake Radiosity Frame:** Bakes a single frame.

**Bake Radiosity Scene:** Bakes a scene file.

**Cache Frame Steps:** Bakes every N frames, with a default of 10.



## Lighting Considerations

Radiosity must consider *global illumination*, which means accounting for all lighting, whether direct or indirect. This includes indirect light provided by the backdrop, sometimes referred to as “sky lighting.”

When the radiosity rays are fired, the color and brightness of the closest element hit by each ray (other than direct light sources, which are accounted for separately) are added into the global illumination for that point. It doesn't matter if a ray hits another diffuse shaded polygon, a luminous polygon, or the backdrop — whatever is hit will be taken into account.

Of course, if the backdrop is blocked by objects, it won't affect shading. For example, in a scene with an *infinite* ground plane object, the lower half of the backdrop gradient won't matter since no rays will ever reach it. Similarly, when shading the floor of an enclosed room, only the areas of the backdrop seen through open doors, windows or skylights will contribute.

Moreover, with gradient backdrops, orientation matters. If the sky is dark blue at the zenith, but bright at the horizon, then the sides of an outdoor object may get more sky lighting than the top. This is because radiosity rays are fired more densely near the direction of the surface normal than around the base of the sampling hemisphere for each shaded point. Thus, light coming in perpendicularly is more important than light coming in at a glancing angle.

## Volumetric Radiosity Command

The Volumetric Radiosity command can be used to control whether volumetrics are taken into account by radiosity rays. This is enabled by default, but can be disabled to avoid potentially long rendering times. You will need to add this command to a menu or shortcut to access.



## Radiosity and High Dynamic Range Images

High dynamic range images (HDRI) contain color and brightness information beyond what is possible in standard 24-bit image formats. One of the most obvious uses for them is with radiosity.



Images Courtesy Darkside Animation

A neat trick is to enclose your scene in luminous polygons mapped with HDRI. In this manner, you can actually light a scene without any conventional lights! You can simulate environments by using photographs with HDRI data and illuminate 3D worlds. You can also do this by applying the HDR image as a backdrop using Textured Environment or Image World.

## Ambient Light

Ambient light, controlled by **Ambient Intensity** on the **Global Illumination Tab (Render Globals>Global Illuminosity Tab)**, is not directly added to surface points that are shaded with radiosity — where radiosity rays originate. However, it is added to points that are shaded without radiosity — where radiosity rays hit a surface. If not for this, a new evaluation point would be spawned at the *hit points* and render time would explode.

Ambient light will still brighten every surface, but only indirectly, after bouncing off other surfaces. Thus it can simulate light that would have come from further radiosity bounces.

As a result, the same exact polygon can be lit in two different ways during the rendering of the same image! The reason for this is that shading is computed on the fly while rendering. Just because a polygon is shaded with ambient light when hit by a radiosity ray doesn't preclude it from being shaded without ambient light in the part of the frame where it's seen directly by the camera — polygons are forgetful and don't *remember* being hit by those rays.

For example, let's say we have a white floor polygon and a red ceiling polygon, 25% ambient light and no direct lights or other objects. If you render this without radiosity, ambient light will show the floor and ceiling as dark gray and dark red polygons. However, with radiosity active, the floor will appear red too.

The floor is red because it no longer gets direct ambient light. Instead, it is lit by radiosity rays that reach out and find the red ceiling — which is shaded with ambient light when hit by these rays — and that red indirect light illuminates the white surface, resulting in red. When the ceiling is rendered, it too is now lit only by radiosity rays that are hitting the white floor (which now gets ambient light as far as those rays are concerned) and that white indirect light illuminates its red surface, so it ends up red too.

## Using Radiosity

"Blotchiness" in areas lit by radiosity occurs when different evaluations come up with different results. If the lighting environment is complex and there are not enough rays to properly sample it, then two or more nearby evaluations can be different enough to cause blotchiness. This can occur, for example, when **Tolerance** is set too high.

Consider a scene lit only by a single luminous polygon. The smaller that polygon is (as seen from the surface being shaded), the larger the number of rays necessary to ensure that at least one of them hits it. If only one of two adjacent radiosity evaluations contains a ray that hits that polygon, then the other evaluation will result in a dark patch on the surface. You can minimise this effect by increasing the *evenness* of the environment — that is, making the luminous polygon bigger or dimmer or using more rays to increase the density of the sampling pattern.

The same principle applies to indirect lighting. A brightly lit area is very much like a luminous polygon as far as radiosity is concerned. For example, when shading a ceiling, more rays are required if the adjacent wall has a few tight spotlights shining on it than if the wall is lit by a broad floodlight. Specular highlights or caustics can also act as tiny indirect light sources that can require a higher sampling density.

Without radiosity, lighting is relatively simple. Basically, it amounts to a small range of light intensities and a linear mapping of the resulting surface brightness into image pixel values. Radiosity, however, is essentially a *physically-based* simulation, with all of the complications that go with it. In the real world, light intensities vary over a huge range, and so it is with radiosity.

Radiosity by its very nature tends to be dim. Ambient light will still be a factor and can help brighten the scene. You may want to process rendered images with a gamma correction filter for radiosity renderings viewed on a computer monitor. This boosts pixel values in shadowy areas, and makes radiosity effects more noticeable. Playing with Image Viewer FP's exposure controls can often reveal detail hidden in an HDRI image such as LightWave produces when rendering.





## Caustics

Generally, a caustic occurs in the real world when light reflects off a curved surface or refracts through a transparent surface so that it is focused on a small area. Light through a wine glass is a good example. With a more complex surface, the caustic can create a random pattern like those seen on the floor and sides of a swimming pool.

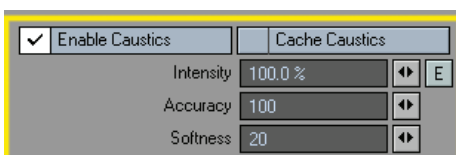


Bottle Image Courtesy Clive Biley, Tag Creative



Copyright Cheng Kai

Activate **Enable Caustics** to render this effect in your scene. **Cache Caustics** saves caustics data for subsequent render passes and frames, which can significantly reduce rendering time. The results can be inaccurate if objects or lights are animated, but this option works particularly well with scenes like a walk-through in which only the camera moves.



**Intensity** is a scaling factor on the brightness of the caustics. If a light shines on a disco ball, casting dots of light around a room, and you want to halve the intensity of the light source without affecting the brightness of the dots, you can just double the caustic **Intensity** to compensate. This parameter does not affect rendering time nor interact with the other two caustics settings.

**Accuracy** (1-10000) determines how many caustic rays are *fired* to compute the caustics. The time needed for the initial rendering pass is directly proportional to the **Accuracy** setting, so you can reduce the setting to speed up rendering. However, if you need to accurately render sharp-edged or intricate caustic patterns, you may need to increase the value. You may also need to increase it if the reflecting or refracting objects in the scene are small compared to their distances from the light source — they are a harder target for the rays to hit.

**Softness** (1 to 100) determines how many nearby caustic rays to take into account when rendering. It affects the speed of the *polygon* rendering pass (as opposed to the caustics preprocessing). The more rays are averaged together, the more blurry the effect. If the caustics are too *noisy*, this setting can be increased. Reducing it will produce sharper caustics, which may require a higher **Accuracy** setting.



NOTE: The default values for Accuracy and Softness should be fine in most cases.



Copyright Evgeny Terentyev

If you do not want specific lights to contribute to the caustics effect, deactivate the **Affect Caustics** option on the **Light Properties Panel**.

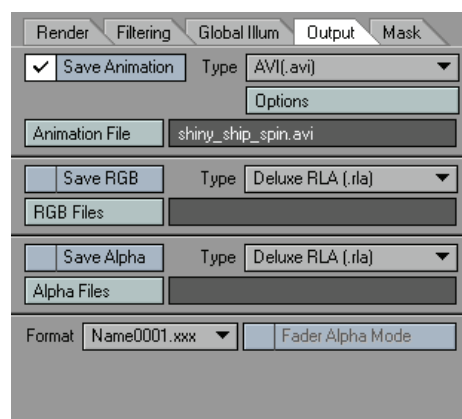


## Radiosity and Caustic Caching Considerations

You should not use the cache options for radiosity and caustics when using distributed rendering on multiple machines. Each machine may have a different cache, which can cause flickering.

Unfortunately, not using caching will probably result in flickering too — even when you render on just one machine. Generally, the most stable method is to render on a single machine with caching active. This assumes, however, that nothing changes during the scene that would affect the lighting (otherwise caching should not be used). Flickering may be less severe if you increase **Rays Per Evaluation** for radiosity or **Accuracy** for caustics.

## Render Globals: Output Tab



This is where you decide just how you want to save your renders when you hit **F10** to render your scene. You can save multiple image types — animations, still frames, alpha images — all at the same time at no extra cost in rendering time.



NOTE: If you are network rendering, you will not be able to save an animation, only still frames.

To choose a save type, just click on the tick box associated with it. This will open a file requester where you can choose a base filename and location. The file extension will be added according to your setting in the **Output Filename Format** drop down menu.

If you wish to change the base filename, just click on the **File Type** button (Animation, Image, Alpha) again. This will open the **File Requester** again and you can change the base name.

You can turn on and turn off any of the save types without losing their settings.

## Save Animation

Once you have chosen a base name for your animation you can choose a type. It's a good idea to save an animation only in addition to, not instead of a file sequence. The reason for this is that in the case of a crash during a render you will have to re-render an animation from the start again, but with an image sequence you will be able to resume from the last frame rendered.



NOTE: A file sequence is easily made into an animation using LightWave. Simply use the sequence as a background image sequence in an empty scene.

If you set the resolution the same as the original frame size and save an animation, you will rapidly convert your still frames into an animation.

## Special Animation Types

Although most of the animation file types are self-explanatory, there are some that may require some explanation.

## Storyboard

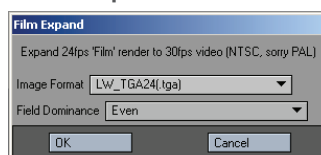
There are two storyboards in the list a 4x one and the standard. They are unusual in the sense that they don't make an animation, but rather a series of images as a storyboard. LightWave renders out each frame and pastes them together in grids of either 4 x 4 or 6 x 5 images, skipping every fifth frame. The standard storyboard makes a grid of 6 x 5 images with each image of 30 frames no bigger than the full individual image size (so a 640 x 480 storyboard frame will contain a grid of 30 images that would normally be 640 x 480). The 4X Storyboard will composite the images together at the size you render at, making for large images if you render at video resolution.



NOTE: If you are using either storyboard function in realistic render mode, it's a good idea to turn off antialiasing and any shaders like Digital Confusion

you may have activated, just so you can make the storyboards quickly.

## FilmExpand



This converts an animation from 24 fps (film speed) to 30 fps (NTSC video speed) by rendering 30 frames for every 24 of the original animation. The interpolation works best when you use Field Rendering (**Camera Properties Panel**). When you select Field Rendering, an **Options Panel** appears where you can select the type of image to save and set the field dominance. It only works for NTSC video, not PAL.

## QuickTime\_Stereo

This is used with the AnaglyphStereoCompose filter for creating animations in the red-blue glasses style.





## QuickTime Virtual Reality Object Saver

QuickTimeVR\_Object is a QuickTime Virtual Reality™ object saver, which adds the ability to pan and tilt the camera inside the animation itself.

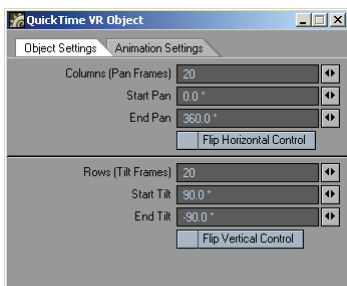
### What is a QuickTime VR Object?

QuickTime VR Objects are QuickTime movies that display an object from multiple views. Special data embedded in an otherwise ordinary movie tells QuickTime how to play through the frames of a movie as a viewer moves the mouse across the image. When the viewer moves the mouse down, the player skips a large number of frames to find one that corresponds to a view of the same side of the object, from a lower position. This can simulate rotating the object in space, if the views are right.

Once a scene with the appropriate camera and/or object movement is created, open up the **Render Globals Panel** and select QuickTimeVR\_Object (.mov) under the **Type** pop-up menu. Click the **Options** button to set the critical parameters for your animation.

### Object Settings Tab

The image below shows the **Object Settings Tab**.



**Columns** (Pan Frames) is the number of frames of horizontal views in the animation. If your view is from 0 degrees at frame 0 to a full 360 degrees at frame 20 (the same view as 0°), then you will have 20 columns (frames 000 to 019). In a typical movie, the row angle will step up after this many frames.

The **Start Pan** and **End Pan** values set the angle range of the horizontal views.

**Flip Horizontal Control** reverses the object's direction of rotation with regard to horizontal mouse movement. In other words, if you drag the object left and it turns to the right, you need to change this setting (or reverse your entire animation!).

**Rows** (Tilt Frames) is the number of frames of vertical views in the animation. In a QTVR Object Movie, the animation moves through all the pan frames in a row before moving to the next tilt frame, and does all the pan views again with that tilt.

If there are 20 pan frames (000 to 019, as above), the tilt angle will change every 20 frames, so sequential tilt values will be at frames 000, 020, 040, 060, etc. This means that your entire animation must be (tilt frames)\*(pan frames) long.



**HINT:** The default Columns and Rows settings (20 x 20 = 400 frames) are more than enough for most uses and you may be able to use lower values to reduce the size of the file.

The **Start Tilt** and **End Tilt** values set the angle range of the vertical views.

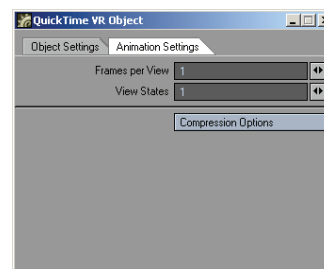


**NOTE:** The Start and End settings may not seem to change simple viewing of the objects, but may be critical if the object is embedded in a larger QTVR world.

Flip Vertical Control reverses the direction of rotation of the object with regard to vertical mouse movement. If you drag the object up, and it rotates down, you need to change this setting.

### Animation Settings Tab

This tab sets some advanced **QTVR** options.



Frames Per View sets the animation loop size that will run from each view. If this value is greater than one, then your animation should hold each pan view for this many frames, as the loop happens. This also means that your animation length must be multiplied by this number. You should not alter the pan or tilt frame counts; QTVR will handle that.

The QTVR object format features an alternate state that can be displayed based on some user input, for example, when you click the mouse. The View States value supports this alternate state, but will also multiply your animation frame count. These extra frames take the form of an entire identical animation of the object in an alternate state, appended onto the first. Currently, the QuickTime player appears to support only two usable states, not clicked and clicked. In theory, higher values could be used.

The **Compression Options** button will invoke the standard **QuickTime Compressor** selection dialog. The choices are many and the differences are subtle. The only relevant warning for **QTVR** is that you should set the compression to use keyframes at every frame, since these movies are not played in a linear fashion.

### To set up a basic QTVR scene:

**Step 1:** Add three null objects to an empty scene, named Pan, Tilt, and Target.

**Step 2:** Parent Tilt to Pan and parent the camera and light to Tilt.

**Step 3:** Set the camera's position to 0m, 0m, -2m.

**Step 4:** Unselect the X and Y motion channels, as well as all the rotation channels. Moving the camera on Z will determine its distance from the actual object necessary for proper framing. All other camera motion is done with the Pan and Tilt nulls.

**Step 5:** Parent Pan to Target, for future convenience.

At this point, you should decide how many rows, columns, view states, and animation loop frames you want. Here, we will use the defaults.



It is best to start on the innermost loop first. Since we're using defaults, with no extra animation loop frames (Frames Per View is 1), the inner loop is the pan motion.

**Step 6:** Create a keyframe for the Pan object at frame 20, with the Heading at -360 degrees.

**Step 7:** In the **Graph Editor**, select both keys and set their Incoming Curve, Pre Behavior and Post Behavior to Linear. This should give a constant heading rotation of one cycle every twenty frames.

If we had chosen to use only one Tilt Frame, our work would be done. We chose, however, to create twenty Tilt Frames, ranging over 180 degrees. This means that each tilt frame should increase enough that the last row is at -90 degrees.

If we simply used, 180 degrees/20 or 9 degrees, the Tilt's pitch would reach -90 degrees at precisely 400, the first frame after our render! Using  $180 \text{ degrees}/19 = 9.47 \text{ degrees}$  will get the tilt to -90 degrees at the end of the 19th row, so our animation will include a pan from directly below as well as directly above. Since there are 20 Pan frames, we must hold each tilt angle for that many frames.

In short, our motion curve should look more like stairs than the perfect line of the Pan Heading curve. To be more specific, Tilt's Pitch Angle curve should be stairs of a height of 9.47 degrees, and a width of 20 frames, going from 90 to -90 degrees. Fortunately for us, this is easily achieved.

**Step 8:** In the **Graph Editor**, set the key at frame 0 in Tilt's Pitch channel to a value of 90 degrees and create a key at frame 20, with a value of 80.53 degrees ( $90 - 9.47$ ). Set the Incoming Curve to Stepped, and the Post Behavior to Offset Repeat. The camera motion should now be complete.

**Step 9:** You can save the scene as a basic QTVR template.

In order to create an interesting QTVR Object, you need something to render. If you load an object that is too large or too small you can move the camera along the Z axis (use the Local Coordinate System).

If the object is off-center, it should be moved. The object should fit in the frame of all the views, so it is important to check the ends and middle of the animation, but only set the Z keyframe position at frame 0.

Try loading the cow object and follow the preceding steps to create your own QTVR animation.



NOTE: There are many ways to set up a LightWave scene to render a QTVR Object Movie. The above procedure is only one method of many.

## Saving Individual Images

You have several choices for saving images, you can either save them at 32-bits to have an included alpha, for those formats that support it; in 24-bit with no alpha or with a separate alpha file (by turning on the Save Alpha image) or if you are working with HDR images, you have the choice – some formats support built-in alphas, some don't (LightWave's native FLX and the SGI 64-bit do; Cineon, Radiance and TIFF LogLuv don't). If you choose to save a separate alpha, make sure it has a different name than your color image!

## Selecting a Filename Format

When you save an RGB or alpha image, LightWave will append a numbered extension to the "base name," based upon the frame rendered. The file naming convention is determined by the **Output Filename Format** pop-up menu setting. "Name" will be replaced with the base name. The number one with the leading zeros (e.g., 0001) indicates the number of digits used for the numerical sequence. The standard PC file extension will replace ".xxx".

For example, with Name001, frame 34 of an animation using the base name Explode would save out as Explode034. Using Name001.xxx, Explode as the base name, and a Targa file format, the 56th frame would save an image named Explode056.tga.



NOTE: The .xxx formats are recommended for general use, since most applications require that you use the proper filename extension.

A filename example appears to the right of the **Save** buttons. The last used Output Filename Format is saved as the default for the next time you start LightWave.



WARNING: If the images may be used on a platform that supports only "8+3" filenames, make sure to keep your base names to a maximum of four or five characters (depending on whether you select a Name001 or Name0001 Output Filename Format).

## Fader Alpha

If you plan to use an alpha channel generated by LightWave along with an external video fader or linear keyer, or a different compositing program, you may need to activate Fader Alpha Mode. Certain switchers can use an alpha image as a fade control. When you select this mode, LightWave computes the saved RGB images and alpha images a bit differently. Transparent items in an RGB image render in a more intensified manner, with rough, aliased edges. Lens flares appear overblown. In alpha images, however, the transparency levels render properly so that the appropriate transparency levels are used when you combine the RGB images with the alpha channel.



WARNING: Do not use this mode if you plan to digitally composite images in LightWave or another digital compositing package that allows for additive compositing.



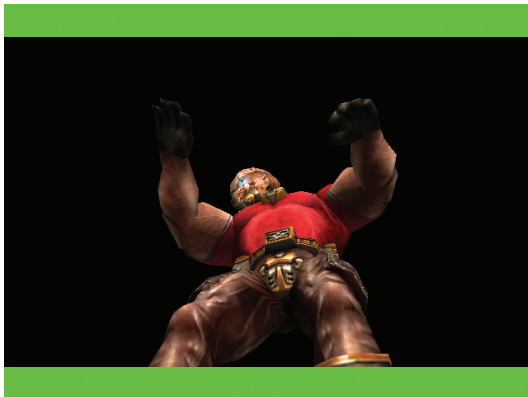
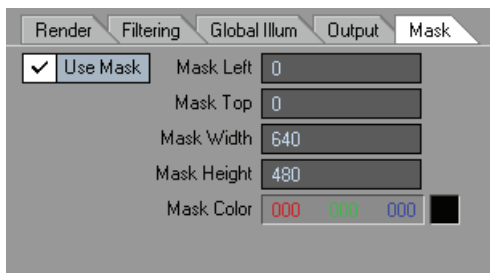
## Render Globals: Mask Tab

### Masking out a region

Using a mask is a little like rendering a limited region, but allows you to define a color for the area outside the region you define. Set the Mask Options in the Render Globals>Mask Tab. Click on the Use Mask button to open up the settings for use. The figures you enter dictate the render area; everything outside it will be the color you choose. You can use this feature to get a letterbox-style effect.



NOTE: You must have Mask Options enabled in the Render Global Panel to use the Mask Options in the Camera Panel.



## Rendering a Limited Region

(default keyboard shortcut **L**)

You can use a *limited region* to mark off a specific rectangular area to be rendered. This is a great feature when you wish to test render an area of the frame without rendering the entire image.



The Limited Region setting is available both on the Render tab and on the Camera Properties panel, and can be called up with the **L** key. Using the keyboard shortcut will switch between three different modes - **Limited Region with Borders**, **Limited Region No Border** and no Limited Region.

When you have Limited Region on, you will see yellow dotted bounding box surrounding your camera view, dragging the lines allows you to resize the limited region you wish to set, while clicking and dragging the **LMB** inside the bounding box allows you to move the Limited Region area around your camera view. To stop editing the Limited Region, choose **Limited Region with Borders**. This renders the area you have selected but puts it onto a black background at the image size you have chosen in the Camera Properties Panel.

### Limited Region No Border

This creates a render at the size you set in the Limited Region setting.

The **Limited Region Bounding Box** has been updated to have handles. They can be enabled/disabled in the Preferences Panel. The handles are represented as small rectangles.

### Memory Considerations

**Limited Region** allocates only enough memory to render the *horizontal* limited region area. If you *stitch* parts of an image together, you can effectively render images that are much larger than those you could render one pass. This is especially useful for high resolution print images or in low memory situations. However, note that some post-processing filters require a full-sized image. In such cases, you may be able to apply those filters to the "stitched" image in an additional step.



## Render Frame

(default keyboard shortcut **F9**)

To render the current frame, determined by the position of the frame slider, choose **Render > Render Current Frame** or press **F9**.



NOTE: Colors on your computer monitor will not match actual NTSC/PAL colors. If possible, always test frames from LightWave on a video monitor prior to final rendering.



NOTE: The Render Status window is a modal window. You may not be able to access options on other windows until it is closed.

## Render Scene

(default keyboard shortcut **F10**)

Select **Render > Render Scene** to render the entire scene using the defined range of frames on the **Render Globals Panel**, or press **F10**. Make sure **Auto Frame Advance** is set as desired!



NOTE: Colors on your computer monitor will not match actual NTSC/PAL colors. If possible, always test frames from LightWave on a video monitor prior to final rendering.



NOTE: The Render Status window is a modal window. You may not be able to access options on other windows until it is closed.

## Render Selected Object

(default keyboard shortcut **F11**)

You can render only the selected object(s) at the current frame by choosing **Render > Render Selected Objects**. Unselected objects can still cast shadows or be seen in reflections on the rendered objects.



NOTE: Colors on your computer monitor will not match actual NTSC/PAL colors. If possible, always test frames from LightWave on a video monitor prior to final rendering.



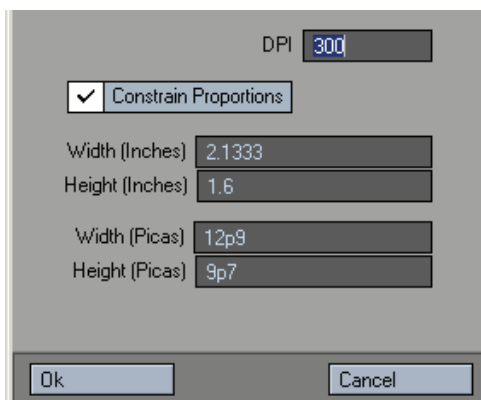
NOTE: The Render Status window is a modal window. You may not be able to access options on other windows until it is closed.



NOTE: If you are using LightWave on a Mac using OSX 10.3.5 or higher, Apple's Exposé might interfere with LightWave's default rendering keyboard shortcuts.

## Print Assistant

The **Print Assistant** will enter width and height details for your render based on inch or Pica measurements at a specified dpi rate.



NOTE: You can also enter print sizes directly into the width and height fields using LightWave's ability to do maths in these fields. For instance, the seemingly complicated sum:  $8.26 \times 300$  gives the width of an A4 page in inches at 300dpi. If you would rather work in metric, the width of an A4 page is 21 cm, so:  $21 \times 300 / 2.54$  will give you roughly the same result (the /2.54 converts the sum into inches). The centimeter value is more precise since an A4 page's size is worked out based on metric rather than imperial measurements.



## Network Render

LightWave allows you to use the processing power of other computers on a network to render scenes. This is called distributed rendering or sometimes a rendering farm. There are a few basic approaches to distributed rendering with LightWave.

ScreamerNet uses Layout's **Network Rendering Panel (Render > Network Rendering)** to control submitting scenes to networked computers running the ScreamerNet process. ScreamerNet can control up to 1,000 CPUs (Note: A single machine can have more than one CPU. Each CPU is counted as one). Each will render a frame from the animation until the scene is rendered.



NOTE: See Chapter 27 - Distributing Rendering section for more information.

## Batch Render on One Computer

You do not actually need a network in order to use ScreamerNet. Just follow the network instructions as if the computer was the node and control machine. It's a little easier since you won't have to worry about shared directories and volume names. ScreamerNet is useful for rendering a series of scene files unattended, and it is most beneficial if you are using a dual-processor system — each processor could be treated as a separate CPU for rendering.

### Troubleshooting

If clicking the **Screamer Init** button doesn't find the other CPUs, go back and start at the Host Machine Setup section.

If images seem to render unrealistically fast and no images are saved:

**Step 1:** Check to make sure that you have full sharing access across the network. You can check this by copying a file at random back and forth across the network.

**Step 2:** If the scene and/or objects were created without taking into consideration the new drive path names, rendering may occur on only the host machine.

The most common cause of ScreamerNet crashing is when too many computers try to write or read their information to/from the Host computers while the host renders.

Do not use ScreamerNet to render on the host machine, but rather use it only as a server where the hard drives are found.

Map a drive from a different computer as drive Y:, for example, and set your scene to save the animations to that drive. The computers don't know that drive Y: is not on the host, just that it's present.

Another problem occurs when the hard drive where you save the images or animation is full.

This problem creates an error in LightWave and on each ScreamerNet node.

## Rendering Without LightWave

The LWSN program has a third option that lets you render a scene without running LightWave. There is no control machine and thus it is not a distributed rendering situation. You must tell the program specifically what to render. The method is run from a DOS prompt using the following syntax (one line):

```
LWSN -3 [-c<config dir>] [-d<content dir>] <scene file>
<first frame> <last frame> [<frame step>]
```

As you can see, you supply the program with the basic information needed to render a scene.

An example would be:

```
LWSN -3 -cD:\Lightwave\Configs -dM:\Newtek Spicegirls.
lws 1 900 1
```

In the example, the program would render frames 1 through 900 of the Spicegirls.lws scene using the Lw.cfg file stored in the D:\Lightwave and using M:\Newtek as the Content Directory.

The configuration file specification is optional, if the Lw.cfg file is in the current directory. Likewise, if the Content Directory is correctly specified in the configuration file, you do not need to give that parameter.



HINT: You can get the syntax for ScreamerNet by simply typing LWSN with no arguments.



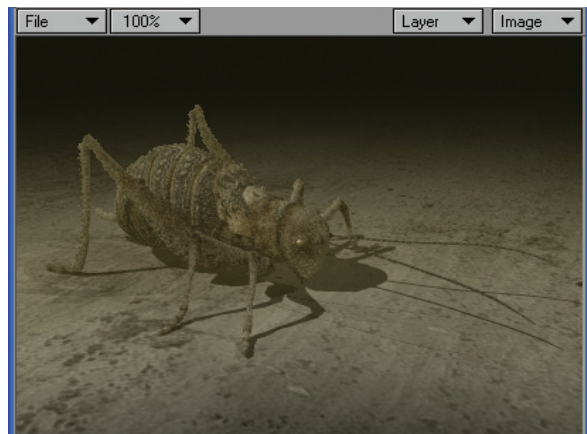
NOTE: See Chapter 26 - Distributing Rendering section for more information.



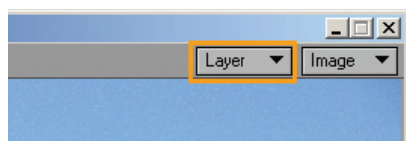


## Image Viewer

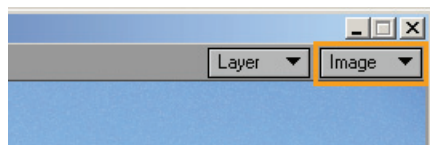
The **Image Viewer** is used throughout LightWave (**Render Display**, **Image Editor**, etc.) to show an image using colors up to the capabilities of your computer's display. Once open, you do not need to close the Image Viewer window. In fact, if you do, any unsaved images will be lost.



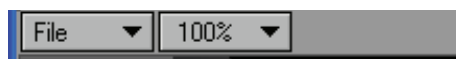
The Image Viewer can sometimes hold multiple images, depending on how it is used by LightWave. In such a case, you can select those images using the **Layer** pop-up menu.



You can choose to see the regular image or the alpha image, if applicable, using the **Image** pop-up menu in the upper-right corner.



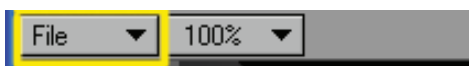
If you drag your mouse pointer over the image, information about the pixel beneath the pointer will appear in the titlebar. The first set of values is the X and Y position. This is followed by RGBA color information where 100% equals 255 for non-floating-point images. For floating-point images, 100% equals 1.



### Titlebar Pixel Information

You can adjust the size of the window by dragging the lower-right corner. Any time the entire image is not fully visible in the viewer window, hold the **Alt** key to scroll around.

## The File Menu

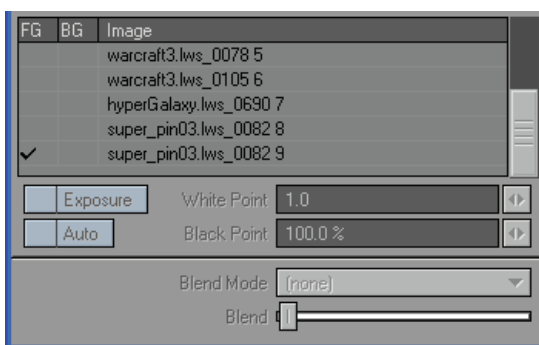


Using the **File** pop-up menu, you can save the current image to a file. Once you select the format type, a file dialog will appear. Make sure you add the appropriate filename extension, if necessary (e.g., Picture.tga). The commands to clear layers are accessed from this menu.

## Image Control Panel

An **Image Controls** menu item also appears on the **File** menu.

On the Controls dialog, you can select a foreground image by clicking in the **FG** column. You can also blend a foreground and background image in real-time using the **Blend Mode** pop-up menu and then dragging the **Blend** slider.



Most of the modes are self-explanatory. The **Difference** mode shows the *difference* between the foreground and background pixels (i.e., foreground minus background). If the **Blend** slider is all the way to the left, the actual difference is shown. Increasing the **Blend** scales up the difference to make it more noticeable. If the foreground and background pixels are very close, the result looks black.



NOTE: The Blend slider will have no effect when using the Alpha blend mode.

If you are displaying a high dynamic range image, you can access special parameters if you activate the **Exposure** option. This allows you to adjust HDR data much like the HDRExpose image filter, but in real-time.

The **White Point** is the input intensity considered to be the hottest white. Anything above this will be the same white in the output. The **Black Point**, expressed as a percentage of the nominal black point (1/255), is the darkest non-black pixel level in the input image that will be preserved. Anything darker will come out black. This control is overridden by the **Auto** option, which uses the image data to compute a black point.



NOTE: When the Image Viewer is the active window, pressing the P key will display/hide the Control Panel.





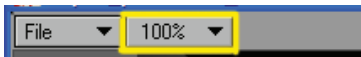
## File Saving

If you are viewing a high-dynamic-range image, and you use the previously described exposure controls to adjust the appearance of the 24-bit display, choosing a file format from the **Save Exposed** sub-menu saves the adjusted image as a 24/32-bit image. The normal **save RGBA** sub-menu preserves all the color information without any adjustments. The full precision of the image (i.e., 24/32-bit, floating-point) is saved (up to the capabilities of the file format).

The **Save Resampled** submenu allows you to save a 24/32-bit scaled version of the image. You can set the **Width** and **Height** independently or if **Lock Aspect** is checked, changing either setting automatically adjusts the other to maintain the same image aspect ratio.

## The Magnify Menu

The **Magnify** menu has several settings that will affect the magnification level of the image. You may also press the plus (+) or minus (-) key to increase or decrease the level of magnification. If the entire image is not visible in the window, holding the Alt key down as you drag on the window will scroll the image around.

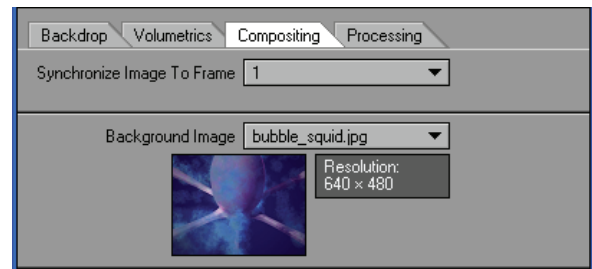


## Compositing Options

(default keyboard shortcut **Ctrl F7**)

### Background Image

The **Background Image** is similar to the gradient backdrop; however, it is always *registered* to the camera. That is, it will always appear in exactly the same position/location no matter which way you tilt or move the camera. You set this option on the **Compositing Tab** of the **Effects Panel**. Background images are considered infinitely distant from the camera. You can never have an object behind a background image, nor can you light a background image or cast shadows upon it.



Background images stretch to fit the Camera resolution and frame aspect that you are using. Make sure to use similar-sized background images if you wish them to match.

You often use background images to merge live action and 3D elements, like creating the illusion that a UFO crashed into the Empire State building. An image of the New York skyline would be the background image and the UFO would be a LightWave object.

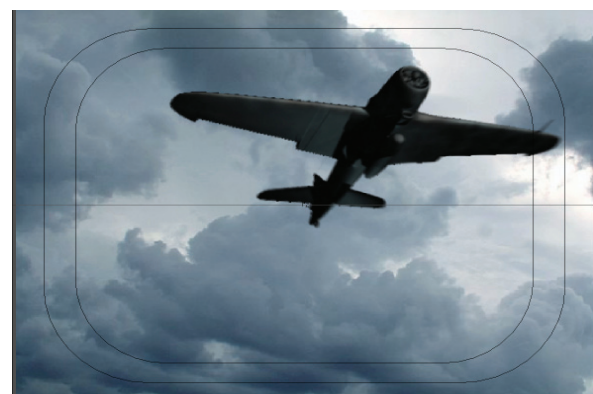


**NOTE:** Background images are visible only through the Camera View.



**HINT:** If you actually need your background image to interact with objects, use the image as a surface color texture-mapped on a flat plane object and place the plane at the back of your scene.

If you set **Camera View Background** on the **Display Options Tab** of the **Preferences Panel (Display > Display Options)** to **Background Image**, you will see the set Background Image in Layout's Camera View. Of course, actually seeing the background in the Layout window is optional. The background image will always appear in the rendered frame.





NOTE: Displaying background in the Layout window, particularly a color one, is processor intensive, so use this feature sparingly.

The whole idea behind traditional image compositing is quite simple: take two or more images, and merge them into a new *composite* image. LightWave lets you do this, but also takes it one step further by letting you throw objects into the mix.

The images can also be a sequence of pictures, so you may use captured video footage as a background for your objects. A simple example would be a modelled UFO moving against a real sky and trees. Images can appear behind all objects, in front of all objects, or a combination of the two with objects showing in-between.

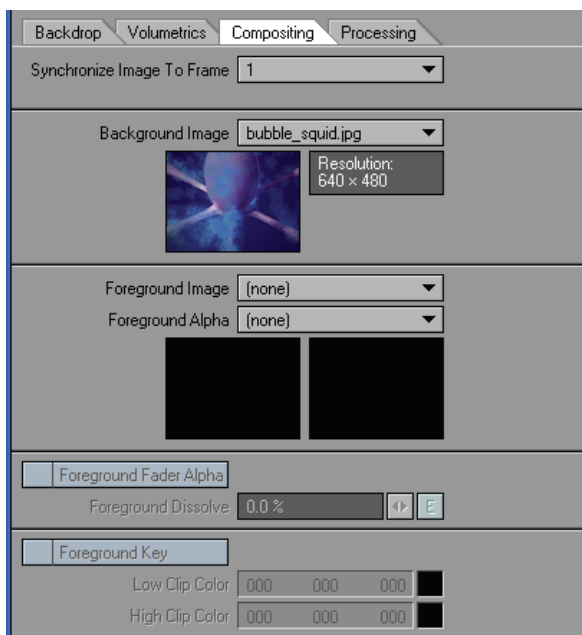
Compositing can be a render time-saver. If you set up a scene with many objects, but only a few are moving, you could render one frame with only the inanimate objects, and then remove all of those objects and render the animated objects against the single frame saved earlier. This is especially useful when the still objects are complicated or involve radiosity, refraction and/or reflection.

## Foreground Images

Placing an image in front of everything may seem like a silly thing to do. However, some options let you cut parts of the image away so you can see through it. You can also make the image partially dissolved or even envelope the dissolve. Dissolving in a black image in the foreground will produce the common *fade-to-black* effect, or reverse the dissolve to fade in from black.

You can make holes in the foreground image based on a defined color range in the image. However, the biggest drawback to simply *clipping* out portions of the foreground image is that you will have hard edges. A color is either clipped or it isn't, so you see either the foreground image or the background image.

Use the Foreground Image pop-up to set the foreground image.



## Alpha Images

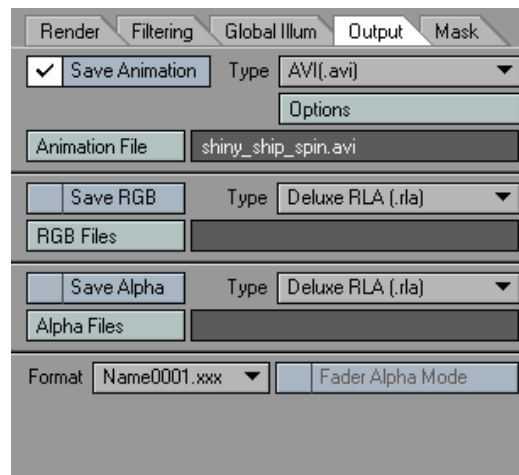
You may also use a special *alpha image* to merge the background and foreground images. This type of alpha image is very different from the type you use to set up a transparency surface texture. LightWave composites the foreground image over any objects or background by *adding* their colors to the foreground image. How much background is added is determined by the alpha image. The darker the area of an alpha image, the more background is added to the foreground. The pseudo-mathematical equation might look like:  

$$\text{Foreground} + (1 - \text{Alpha}) * \text{Background}$$

If you used the same exact image for both the background and foreground images, plus a solid black image as the alpha image, you will receive a final rendered image where every pixel is twice the color value it was. This results from the background image being completely added to the foreground image.

## Creating Alpha Images

Generally, alpha images will be generated when you render a scene to create the foreground images. When you select **Save Alpha** on the **Render Globals Panel's Output Files Tab (Render > Render Globals)**, LightWave will generate and save an alpha image in addition to the normal RGB output. The alpha image will be composed of greyscale values representing the opacity of any objects that were rendered in the scene.



An object that contains no transparent surfaces will be rendered as solid white. Transparent surfaces will render in some shade of grey depending on how transparent they are. One hundred percent transparent surfaces render as black. A 50-percent transparent surface will render as 50-percent grey. Using object dissolve, antialiasing, motion blur, and so on, will also give you values other than white in an alpha image. Any background (image or colors) will be black in the alpha image, as will any additive effects such as glow or lens flare.



Image and Alpha



Since glows and lens flares are additive effects and are assigned a value of black in an alpha image, glows and lens flares in the actual foreground image will simply have the background values added, so they will appear brighter where the background is a value other than black.



**HINT:** Generally, due to LightWave's additive compositing method, foreground images are created using a solid black backdrop. This allows the composited background to show through unaltered when it is mixed 100 percent with the foreground.

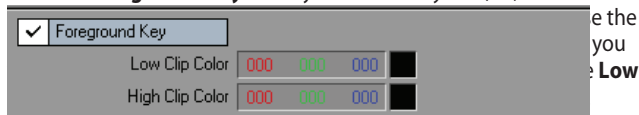
## Foreground Fader Alpha

What happens if you want to composite a foreground element on top of objects or a background, but the foreground image was not rendered over black? In this case, LightWave provides a **Foreground Fader Alpha** button that can be selected. In this mode, LightWave will ignore any areas of the foreground image corresponding to black areas in the alpha image. These areas of the foreground image will be faded away to nothing so you see 100 percent of the background instead.

When using **Foreground Fader Alpha**, glows and lens flares will not be added to the final rendered image (their corresponding alpha is black) unless you are using a different alpha image containing non-black areas in those locations. Additionally, antialiased object edges most likely will stand out because they contain bits of color from the original non-black background.

## Foreground Key

Activate **Foreground Key** when you want to *key out* (i.e., not render



The **Low Clip Color** is the color value for the darkest color value that will be keyed out. **High Clip Color** is the brightest color value that will be keyed out.

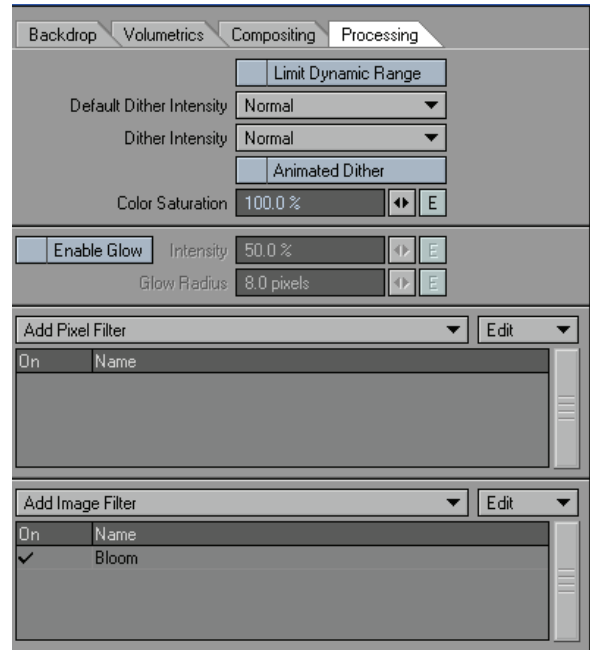


**HINT:** If you want to create a giant space battle and lack the RAM to hold all of the objects and image files, you could composite layers of ships and achieve the same results. This is, in fact, how some broadcast TV shots were done using machines with only 32MB of RAM in the early days of LightWave.

## Image Processing Options

(default keyboard shortcut **Ctrl F8**)

The **Processing Tab** on the **Effects Panel** contains functions that apply effects to the rendered image. Choose **Window>Image Processing Options** to directly bring up the **Processing Tab** of the **Effects Panel**.



## Limit Dynamic Range

**Limit Dynamic Range** clips the pixel color components of each rendering pass at 1.0, improving the antialiasing of extremely bright areas. This option should not be used with filters or image savers that expect high dynamic range data.

## Dither Intensity

*Dithering* blends two colors to simulate a third color between them, forming a more realistic blend. **Dither Intensity** lets you set the amount of color blending used by LightWave when rendering an image. Even with 24-bits of color data, it is possible to perceive color banding where distinct changes in color or brightness occur within otherwise smoothly ramped colors. **Off** removes all dithering, and you will probably experience some color banding. **Normal**, the default setting, reduces banding to the point where it nearly disappears. **2x Normal** increases the dithering even further, which may be useful for high-end systems that still retain some appearance of banding in the final image. **4x Normal** boosts dithering so that the resulting image looks more grainy, like film, which may be a desirable effect (especially when used with **Animated Dither**, below).

## Animated Dither

Select **Animated Dither** to change the dithering pattern used from one frame to the next. This ensures that dithering is randomly placed, so there is no apparent pattern to the dither blend. With a **2x Normal** or **4x Normal Dither Intensity**, this can be used to approximate the randomness of film grain moving through an image.

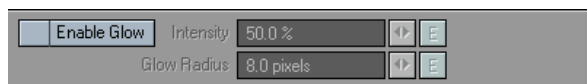


## Color Saturation

**Color Saturation** lets you control the amount of color in a scene (or in an animation, if using an envelope). **Saturation** at 100% is a normal, full-color setting, while saturation at 0% is a black and white setting.

## Glow Settings

When **Enable Glow** is turned active, LightWave can add a glow around surfaces with a (surface) **Glow Intensity** above 0%. Use the controls below to set up the amount of glow you wish to add to all such surfaces.



**Glow Intensity** sets the brightness of the glow, starting from the edge of the surface itself and fading away from there. **Glow Radius** sets the distance (in pixels) that the glow extends from the edge of a glowing surface. Note that different **Resolution** settings will cause dramatically different results.

## Normal Buffers

The Image Filter and Pixel Filters now can access the normal buffer.

There are 3 new buffer values:

LWBUF\_NORMAL\_X

LWBUF\_NORMAL\_Y

LWBUF\_NORMAL\_Z

From the Effects Panel/ Processing Tab in Layout, the image filters:

Render Buffer View

Render Buffer Export

...now have the option to view or save the normal buffers.

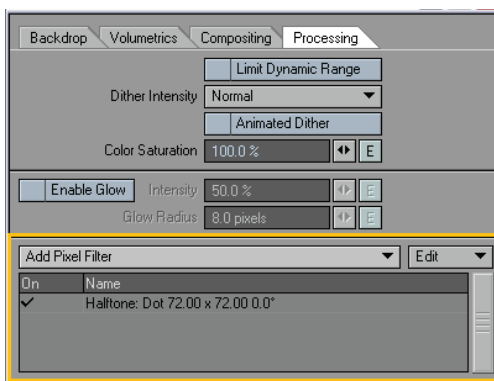
To select the buffers to view or export, load Render Buffer View, make sure it is selected, and from the Edit dropdown select Properties

To select a file type and set the filename for export, load Render Buffer Export, make sure it is selected, and from the Edit dropdown select Properties.

On the Source dropdown menu, select Normal, and set the desired output settings.

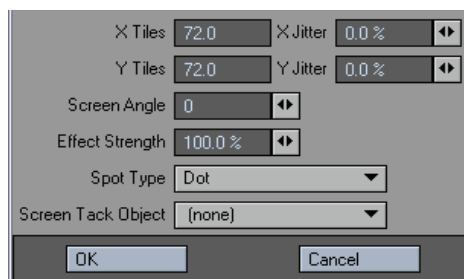
## Image Processing: Pixel Filters

Pixel filters let external applications affect LightWave's rendering engine. Filters in this class can be affected by motion blur and other sub-frame operations during the render rather than as a post process, as image filters are.



## Halftone

In print, halftone screens are made up of dots that control how much ink is deposited at a specific location. Varying the resulting dots' size and proximities creates the illusion of variations of grey or continuous color.

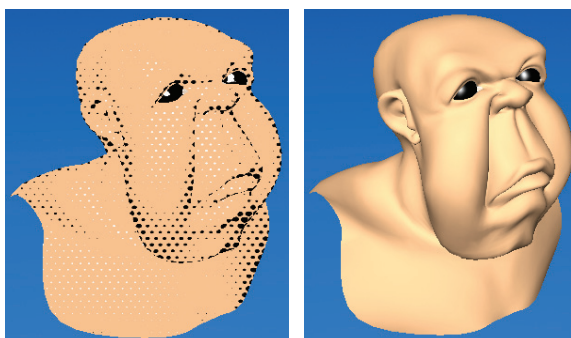


The **X Tiles** and **Y Tiles** values determine the number of possible horizontal and vertical dots. If you would like to randomise either of these settings (throughout your animation), use the corresponding **Jitter** fields.

In the photography world, this type of effect is achieved with a (physical) screen that breaks up the image into dots. Think of the **Screen Angle** setting as the rotation of that screen. It controls the angle of the dots.

You can control the overall amount of effect by adjusting the **Effect Strength** setting. Settings below and above the default of 100% are allowed.

You can change the pattern by changing the **Spot Type**. Specify and animate a **Screen Tack Object** to animate the pattern position.



Left: With Halftone, Right: Without Halftone

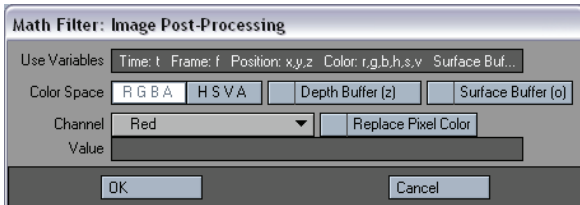


## LW\_Hypervoxels and LW\_Hypervoxels\_Doubler

These two image pixel filters are Legacy tools for rendering older LightWave Scene files that use older versions of **HyperVoxels**.

### Math Filter

This pixel filter allows you to use a mathematical formula to adjust the color in your render.



### SasLite

**Sasquatch Lite** allows you to quickly and easily create a variety of effects such as grass, fur, and hair on your objects. **Sasquatch Lite** uses its own rendering engine to render the hairs quickly and combine the results seamlessly with your LightWave objects. **Sasquatch Lite** is a very simplified version of the commercial plugin *Sasquatch* from Worley Laboratories.



NOTE: See the discussion on the **Sasquatch Lite** displacement plugin located in the **Sasquatch** Section in Chapter 14: Object Properties: Deform tab for more information..

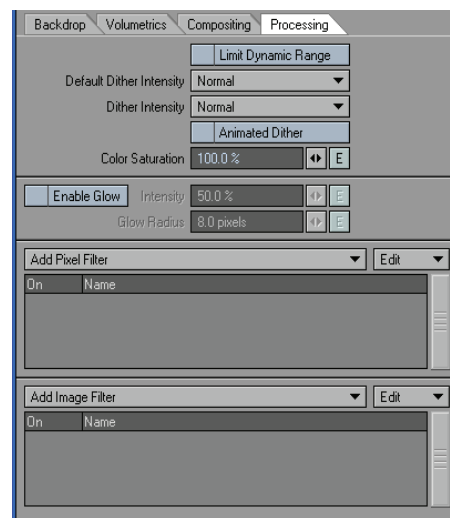
### Steamer and Steamer\_Doubler

These two image pixel filters are Legacy tools for rendering older LightWave scene files that use **Steamer**.

## Image Processing: Image Filters

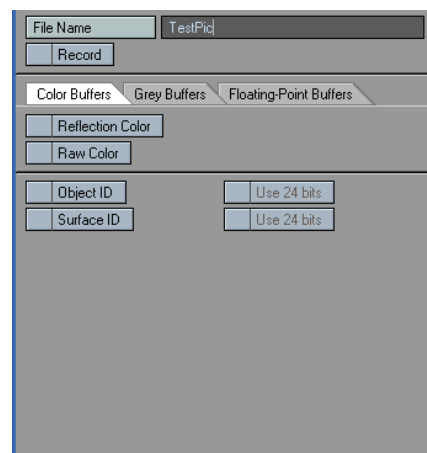
Image filters, unlike Pixel filters in the previous section, act on an image as a whole. They can be used for a variety of things, from saving out a layered Photoshop file to achieving a light bloom effect on your render, or making it into a sepia or monochrome image. Some of the filters can be applied directly to images you use within LightWave — for instance, full precision blur can be used to soften black and white images intended for a **Bump Map**.

Image filters are a post-process effect. This means that they get applied only once all rendering has been performed. Some of them, such as Digital Confusion, by their very nature need your picture to be rendered as one segment otherwise nasty banding effects can occur.



### Aura 2.5 Export

The Aura 2.5 Export image filter has access to all the different internal render buffers and can export them to Aura. Aura 2.5 can then take this information and manipulate it with its many features and options.

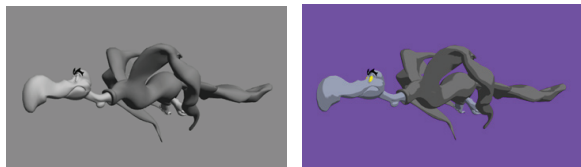






## Black & White

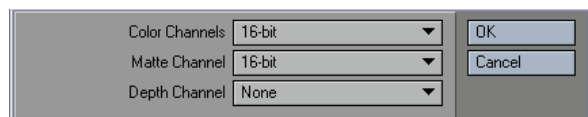
Creates a Black and White version of your render. Double clicking on the item in the list will bring up the Black and White panel where you can alter the gamma correction for the contrast levels.



Left: Black and White Image Filter Applied, Right: Normal Render

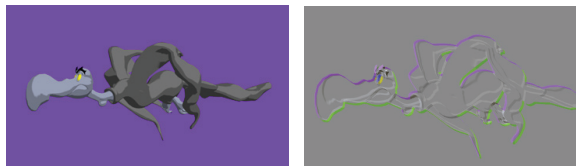
## Deluxe RLA

This image filter allows you to set the format for the color, matte and depth channels for the RLA format.



## Emboss

Emboss gives a 3D shadow effect to the image. It works by taking a pixel on one side of a specific pixel's center, and subtracting one on the other side from it. Pixels can get either a positive or a negative result that dictates whether they will be bright or dark.



Left: No Filter, Right: Emboss Filter

## Emboss Black and White

Basically the same function as above, but the image is converted to greyscale.



## Flare2Alpha

This image filter will add lens flares and certain other volumetrics to the alpha of your rendered image as opaque objects so that they get composited in. Normally lens flares are a post processing addition to the render and therefore they don't get figured into the alpha channel.

## Gamma

This is a Legacy filter. Gamma adds a gamma of 2.2 to your image brightening it a little. This plugin is devalued since LightWave has had the ability to render full precision images and shouldn't be used. Instead use Full Precision Gamma.

## Anaglyph Stereo: Compose

This filter allows you to recreate the kinds of stereo images used with glasses with red and blue lens. By choosing this image filter and also setting up stereoscopic rendering in the **Camera Properties Panel** in the **Stereo** and **DOF Tab** at the bottom of the window, your render will have the two channels, one for each eye, combined into a single red/blue image.

Using the Quicktime Stereo animation type in **Render Globals** will allow you to save a stereoscopic animation. You need to render at least two images to see the stereoscopic ones.

## Anaglyph Stereo: Simulate

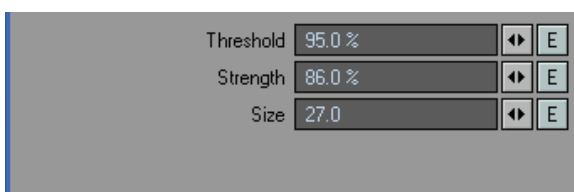
For cases where you want the look of the stereoscopic effect, but don't want to render two images, or you just want to see what effect the stereoscopic camera in LightWave is capable of, there's the Simulate filter.



## Bloom

This filter imitates the way that sometimes too much light is reflected back into the camera from a brightly lit scene. The effect is often seen when photographing or filming shiny metal surfaces or water.

Sometimes light reflections in the real world are so bright that too much light will enter a camera lens, over-saturate areas, and create a glare or a glowing halo. Shiny metallic objects like cars and water often exhibit this phenomenon. Bloom will mimic this effect.



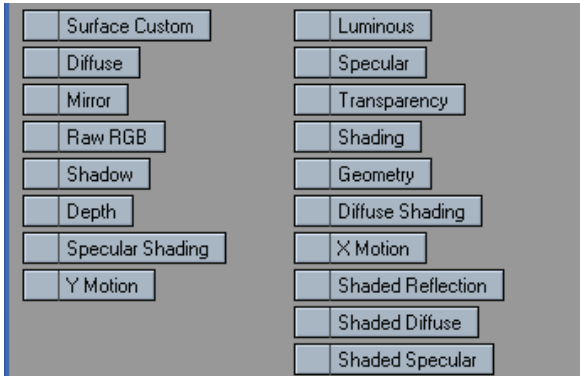
The setting for this filter allows you to set a **Threshold** for the effect – how bright a pixel has to be before it blooms. **Strength** is how bright the bloom effect is compared to the pixel it is overwriting and **Size** is the size of the bloom brush that is used to replace a pixel rendered at a 640 x 480 resolution. If you use a size that looks correct on your 320 x 240 test render, it will still look the right size on your 2000 x 1200 final print-resolution render.



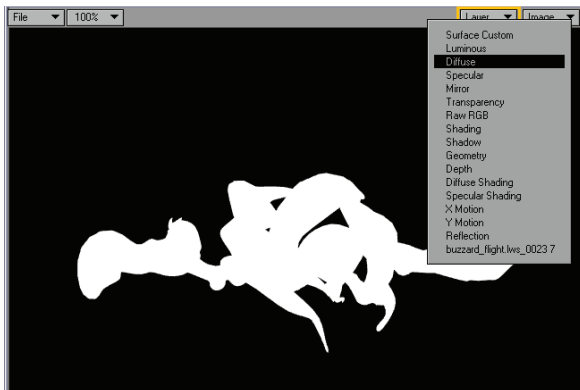


## Render Buffer View (Post-processing only)

This filter makes the selected internal buffers visible as separate images when you use the Image Viewer.

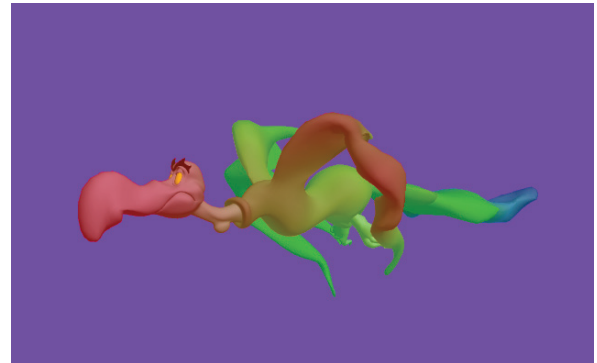


Use the **Layer Drop Down** list on the **Image Viewer** to choose the internal buffer you would like to view.



## Chroma Depth

The Chroma Depth filter makes a stereo image for use with Chroma Depth glasses (see [www.chromatek.com](http://www.chromatek.com)). Basically, color determines the apparent depth in the image.



*Render with Chroma Depth Applied*

This filter recolors the scene based on the Z-depth; it spreads the spectrum of colors from the Near Z Distance to the Far Z Distance. Blend dissolves the Chroma Depth image in with the regular colors to make the depth effect subtler.



Auto-Scale finds the actual near and far for you when it renders — you can render once with this option on and the calculated values will appear in the fields.

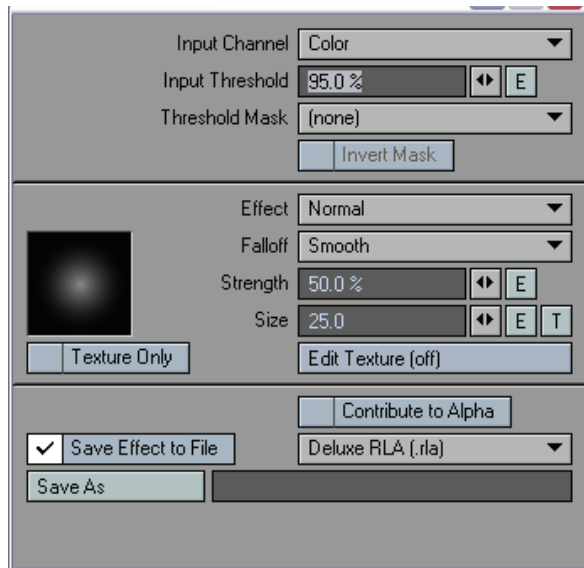


**NOTE:** Use the Bloom filter for a simple over-exposure effect. Use Corona if you need more control over your bloom.



## Corona

This is a more fully featured version of Bloom. With it you have a lot more control over what causes blooming. In essence, control is broken down into which input channels cause blooming, how the bloom takes place and some additional abilities not found in Bloom.



### Input Settings

**Input Basis** lets you choose which internal buffer to read for the bloom effect, so you can bloom on specularity, bloom on diffuse, and so on. The **Input Basis** acts like a trigger: when its value exceeds the **Input Threshold**, the effect is applied to that pixel.

**Color** uses raw pixel values — essentially any pixel on screen that is bright enough gets bloomed. **Alpha** uses alpha's pixel values — 0 to 100% for the image. **Specular Shading** uses 0 to 100% of surface specularity as *shaded* during the rendering. This varies over a given surface and is different from the Specular surface channel, which is uniform over a surface. **Diffuse Shading** is similar, but uses the diffuse surface property.

**Geometry** uses the normal of object surfaces, where 100% indicates that the normal points at the camera. **Inverse Geometry** is similar, but 100% indicates that the normal points perpendicular to the camera. These are easily demonstrated using a sphere. For **Geometry**, the center of the ball would trigger Corona, while **Inverse Geometry** would result in the effect along the edges.

**Special** uses the surface Special Buffer feature on the **Surface Editor**. The value of the Special Buffer is compared against the threshold and when it exceeds that value, the Corona filter is applied.

The input can also be *masked* to skip areas of the input altogether. **Threshold Mask** is basically an alpha channel. Brighter areas will be susceptible to the mask, while darker areas will not.

### Effect Settings

The **Effect** pop-up menu selects your blending mode. **Additive** yields very hot (white) results where triggering pixels are closely grouped. This is useful for, say, obtaining the look of metal being heated. The center of a block of metal will become super hot while the edges do not. **Normal** is similar to **Additive**, except in the case of heated metal, the effect at the center and the edges tends to grow more evenly. **Maximum** takes the maximum of contributive pixels. This yields an effect like applying balls of cotton to brightly colored particles, whose effects start to merge as the particles become closer to each other.

The **Falloff** pop-up menu lets you select how the bloom brush falls off. The preview window will give you an idea of what the falloff will look like.

**Strength** is the strength of the *brush* compared to the source pixel. **Size** is the radius in pixels of the brush at a 640 by 480 resolution. If the resolution is different, the brush will be adjusted so that the effect always looks the same at different resolutions.

The **Edit Texture** button can modulate the color of the bloom brush with a texture.

When **Texture Only** is inactive and **Edit Texture** is off, the effect uses the color of the original image. If texture color is available, the effect uses the color of the original image plus the contribution of the texture. When **Texture Only** is active and there is also a texture color, the effect uses the value of the texture only.

### Other Settings

The Corona filter will affect your alpha channel if you activate the **Contribute to Alpha** option.

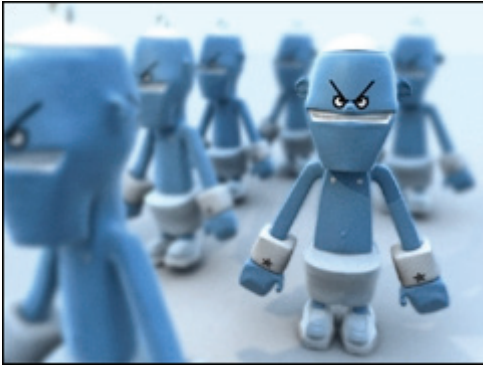
Use the **Save Effect to File** option to save just the corona effect to an image file when rendering. Note you must also choose a file format and set a filename.

When you use gradients with Corona, you will have additional options for the **Input Parameter**. These options let you customise how the corona effect is applied. For example, the size or intensity of the effect can grow or diminish based on an object's proximity to another object, center of the image, and so on.

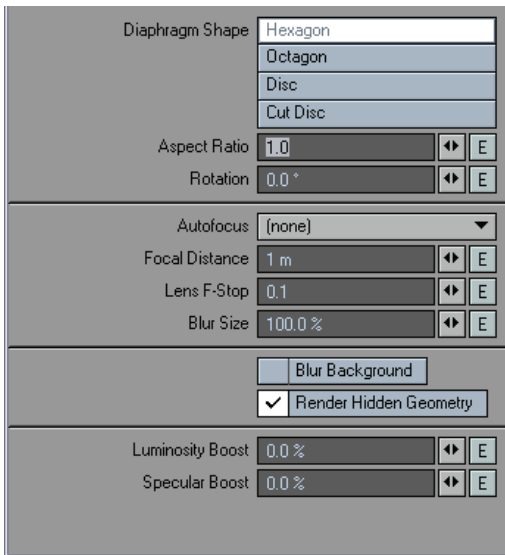
You can use the standard Preset Shelf if you want to save and recall settings.



## Digital Confusion *(Post-processing only)*



LightWave's built-in Depth of Field effect (**Camera Properties Panel**) adjusts which pixels the rendering camera considers in and out of focus. The Digital Confusion image filter creates the same effect using similar controls, but offers several extra features. Adjusting the **Focal Distance** and **Lens F-Stop** still easily controls the range of focus, but added options include camera lens parameters, auto-focusing, and ways to fine-tune surface properties.



Since Digital Confusion is added during the antialiasing rendering pass, it will respect and can actually improve oversampling methods like motion blur and Depth of Field effects. However, Adaptive Sampling (**Camera Properties Panel**) may not function correctly with this filter. In this case, you should use an **Enhanced Antialiasing** setting (**Camera Properties Panel**).

The four **Diaphragm Shape** settings, **Hexagon**, **Octagon**, **Disc**, and **Cut Disc**, determine which pattern Digital Confusion will use when "defocusing" the rendered image. These settings correspond to the actual shape of the camera lens used during this effect. The **Aspect Ratio** and **Rotation** angle of the camera lens can also be adjusted to create even more specialised effects.

When using LightWave's built-in Depth of Field controls, it is sometimes difficult to keep a moving object in focus. Now, instead of using envelopes to animate the focal distance, with Digital Confusion you can simply select a reference object from the **Autofocus** pop-up menu and the proper focal distance will be computed automatically. This reference object can be either the (target) geometry in the scene, or a null object used to dynamically adjust the focus. When an object is selected, the **Focal Distance** field becomes disabled.

The **Focal Distance** setting represents the distance from the camera to the point in space that is in focus. Objects that fall either in front of or behind this area will be rendered out of focus. Just how far out of focus is determined by adjusting Digital Confusion's **Lens F-Stop** setting. By changing this value, you are adjusting the radius of the area that is in focus. For this value, the smaller the **Lens F-Stop** size, the smaller the in-focus area will be.



NOTE: For more information on using the Depth of Field controls, refer to the Depth of Field section of the manual, in Chapter 16: Camera Basics.

The **Blur Size** setting acts as a multiplier to Digital Confusion's "defocusing" effect. Adjusting this control is similar to adjusting the **Lens F-Stop** setting, but instead of changing the size of the in-focus area, it determines the amount of blur these pixels will receive. By entering a **Blur Size** value of 50%, the area out of focus, defined by the **Focal Distance** and **Lens F-Stop** setting, will receive only half the computed blur.

If a background image is used in your scene, you may want to activate **Blur Background** to blur it. However, because the background image is at an infinite distance from the camera, it will always receive the maximum blur amount and will result in much longer rendering times. A more efficient solution is to simply blur the background image in a paint program, or use a blur filter in LightWave's Image Editor processing tab.

The **Render Hidden Geometry** feature forces LightWave to ray trace any geometry behind objects in case they go transparent when being blurred. This is a much more accurate representation of the depth of field effect, but can increase rendering times. In multi-pass rendering and compositing, it may be acceptable to not activate this feature, but normally it should be activated.



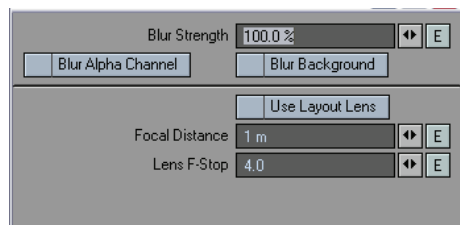
NOTE: Double-sided geometry will not work correctly with the Render Hidden Geometry activated.

Sometimes defocusing the rendered image can cause the effect of surface specularly or luminosity to diminish unacceptably. To offset this effect you can adjust the **Luminosity Boost** or **Specular Boost** multipliers located at the bottom of the panel. Any pixels rendered with these surface properties will have their intensity adjusted accordingly.



## Depth-Of-Field Blur (*Post-processing only*)

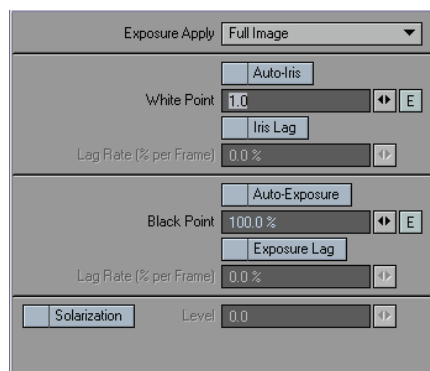
This filter lets you add a depth of field effect that is based on a fast image filter, without requiring multi-pass antialiasing like normal depth of field does (**Camera Panel**). You can adjust the overall strength of the blur, as well as independently choose whether to blur the alpha channel and background.



See the discussion on normal Depth of field (DOF) for information on the **Focal Distance** and **Lens F-Stop** settings. You can also activate the **Use Layout Lens** option to use the **Camera Properties DOF** settings.

## Exposer

This filter normalises high dynamic range (HDR) images for using as **Image Maps** and HDR output for displaying in applications that are not HDR-savvy. The intensity mapping is non-linear, similar to the light response process in the human eye.



This filter processes the HDR output created by radiosity and renders into better-looking, brighter pictures. It does this without impacting the accuracy of the lighting simulation, which can happen if you add ambient light or crank up lights unrealistically. It is really an essential part of the camera simulation, for a perfect digital camera. (The Virtual Darkroom filter is similar, but more complex. It simulates the two-stage process of film response to light, and print emulsion response to projection through the film negative.)

Although you can add this filter on the **Image Editor**, it is of limited use there and more useful as an Image filter on the **Processing Tab** of the **Effects Panel**. This is mainly because most images you load are not HDR images, so pre-processing is not necessary and normal gamma should probably be used, if necessary. Moreover, if you do load an HDR image, it's probably because you want the extra data. (Using the HDR Exposure filter will eliminate some, if not all, of the extra data.)

The Input Dynamic Range is an informational display showing the High and Low pixel-intensity values encountered in the last processed image. Note that when the panel first appears, this information is not yet known.

If you do not want the filter applied to the Full Image, set the Exposure Apply pop-up menu to Foreground to apply it only to scene items or Background to affect only the background (i.e., where your alpha would be black).

The White Point is the input intensity considered to be the hottest white. Anything above this will be the same white in the output. This control is overridden by the Auto-Iris option, which sets the white point based on the actual input image data. Adjusting the white point is similar to cranking down an iris on a film camera to limit how bright parts blow out in a photograph.

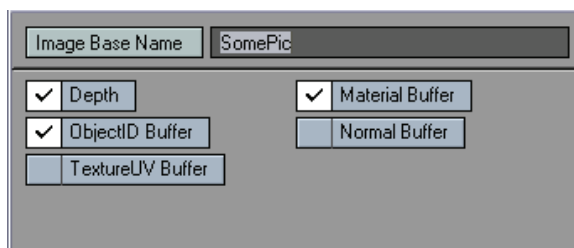
The Black Point, expressed as a percentage of the nominal black point (1/255), is the darkest non-black pixel level in the input image that will be preserved. Anything darker will come out black.

The Auto-Exposure option overrides Black Point by using the actual image data to determine a black point in the incoming data. Lowering the black point is similar to increasing the exposure time for a photograph.

Once these values are set, the filter translates the incoming image intensity — in a process very similar to gamma correction — so that the darker colors get more of the output range than the brighter colors. In other words, the filter changes the spacing of intensity levels so more levels are devoted to low intensity, darker details.

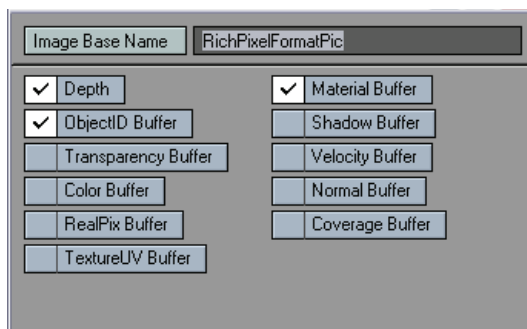
## Extended RLA Export (*Post-processing only*)

This filter saves images in the *Extended RLA* format, popular for 2D/3D compositing work. The image includes optional depth buffers, as well as masks for which object or surface (material) a pixel came from. Enter the **Image Base Name** in the field or use the **File Requester** button.



## Extended RPF Export (*Post-processing only*)

This filter saves images in the *Extended RPF* format, popular for 2D/3D compositing work. The image includes a few more options than the Extended RLA. Enter the **Image Base Name** in the field or use the **File Requester** button.



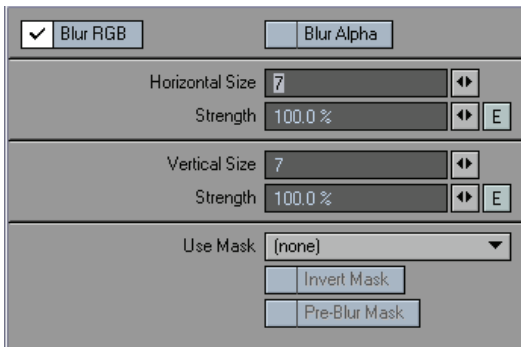


## Full Precision Blur

This filter will soften an image by blurring it. Change the **Size** values to increase the amount of blur horizontally or vertically. The **Strength** settings determine the amount of the effect. You can also choose whether to affect the RGB (color) and/or alpha data.

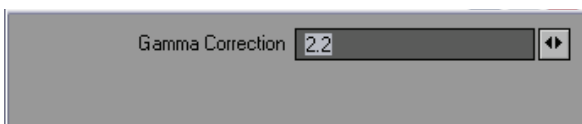
You can use the **Rendered Alpha** (channel) or a **Special Buffer** as a mask using the **Use Mask** pop-up menu. If you want the mask computed prior to the blurring, check the **Pre-Blur Mask** option, otherwise the mask accounts for any blurring. To reverse the mask, check **Invert Mask**.

The **Special Buffer** setting on the **Advanced Tab** of the **Surface Editor** can have a value from 0 to 1, with 0 meaning no blur and 1 meaning full blur.



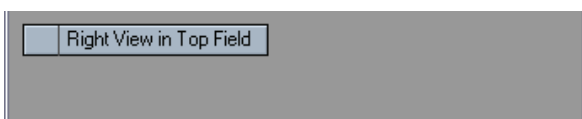
## Full Precision Gamma

Display devices have a non-linear relationship between pixel values and physical light intensity — they do not excite the display phosphors linearly. This non-linearity must be compensated for to correctly reproduce intensity. The **Gamma Correction** value is the exponent value in the correction formula and determines how to convert pixel values to light intensity. The default, 2.2, is a common value used on images bound for video, but is not necessarily the optimum value.



## Field Stereo

Rather than using red/blue glasses to create a stereoscopic animation, you can use LCD shutter glasses that work with the fielded nature of TV screens to work their magic. Field Stereo allows you to render separate images for each field so that you get an effect of depth from the separation between the fields. Don't forget to turn on Stereoscopic rendering and Field Rendering in the **Camera Properties Panel**.

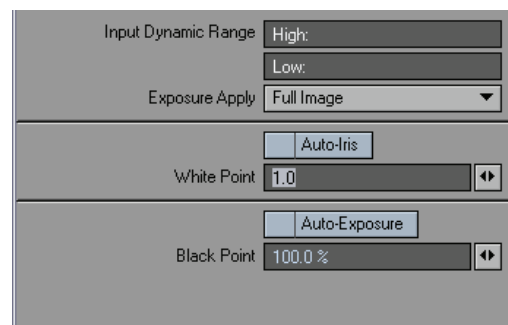


## HDR Exposure

This filter normalises high dynamic range (HDR) images for using as **Image Maps** and HDR output for displaying in applications that are not HDR-savvy. The intensity mapping is non-linear, similar to the light response process in the human eye.

This filter processes the HDR output created by radiosity renders into better-looking, brighter pictures. It does this without impacting the accuracy of the lighting simulation, which can happen if you add ambient light or *crank up* lights unrealistically. It is really an essential part of the camera simulation, for a perfect digital camera. (The Virtual Darkroom filter is similar, but more complex. It simulates the two-stage process of film response to light, and print emulsion response to projection through the film negative.)

Although you can add this filter on the **Image Editor**, it is of limited use there and more useful as an Image filter on the **Processing Tab** of the **Effects Panel**. This is mainly because most images you load are not HDR images, so pre-processing is not necessary and normal gamma should probably be used, if necessary. Moreover, if you do load an HDR image, it's probably because you want the extra data. (Using the HDR Exposure filter will eliminate some, if not all, of the extra data.)



The **Input Dynamic Range** is an informational display showing the High and Low pixel-intensity values encountered in the last processed image. Note that when the panel first appears, this information is not yet known.

If you do not want the filter applied to the **Full Image**, set the **Exposure Apply** pop-up menu to **Foreground** to apply it only to scene items or **Background** to affect only the background (i.e., where your alpha would be black).

The **White Point** is the input intensity considered to be the hottest white. Anything above this will be the same white in the output. This control is overridden by the **Auto-Iris** option, which sets the white point based on the actual input image data. Adjusting the white point is similar to cranking down an iris on a film camera to limit how bright parts *blow out* in a photograph.

The **Black Point**, expressed as a percentage of the nominal black point (1/255), is the darkest non-black pixel level in the input image that will be preserved. Anything darker will come out black. The **Auto-Exposure** option overrides Black Point by using the actual image data to determine a black point in the incoming data. Lowering the black point is similar to increasing the exposure time for a photograph.

Once these values are set, the filter translates the incoming image intensity—in a process very similar to gamma correction — so that the darker colors get more of the output range than the brighter colors. In other words, the filter changes the spacing of intensity levels so more levels are devoted to low intensity, darker details.



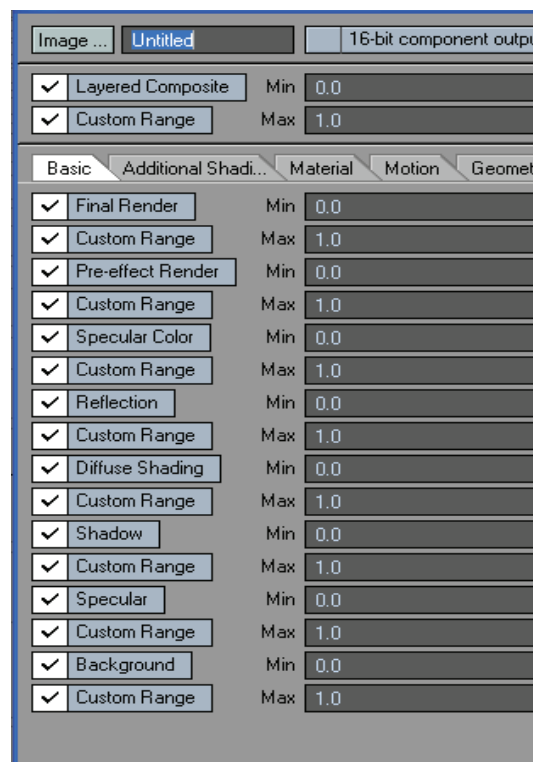


## Lscript and LScript/RT

Both allow you to run an LScript on a finished image.

## Photoshop PSD Export

The Photoshop PSD Export image filter is an image saver masquerading around as an image filter. Being a filter allows it to have access to all the different internal render buffers. When you render a frame, a sequentially numbered PSD file is saved with each of the selected buffers placed into their own channel. (In Photoshop 6, select the **Channels Tab**, between the **Layers** and **Paths Tabs**.)



### Basic Tab

**Final Render:** Represents the final output from the LightWave render.



Note: Final Render is actually several filters in one, all of which will be activated when you look at the PSD file. The following additional filters will be available in the PSD file: Effect (-), Effect (+), Surface, Diffuse Lighting, and Refraction.

**Pre-effect Render:** Represents the render before effects, such as lens flare, are applied.

**Specular Color:** Represents the specular color options of a surface.

**Reflection:** Represents the reflection channel for a surface.

**Diffuse Shading:** A single channel buffer, indicates the amount of light reflected toward the camera from a surface, including the effects of shadows cast onto the surfaces.

**Shading:** A single channel buffer, represents the lighting effects for shadows, as well as specular and diffuse.

**Specular:** A single channel buffer, indicates the value of the specular attribute of a surface.

**Background:** Represents the channel buffer when options in the Backdrop panel are activated, such as Gradient Backdrop or Image World.

Examples of various filters (Render from the Night.lws scene, included with the LightWave content.):

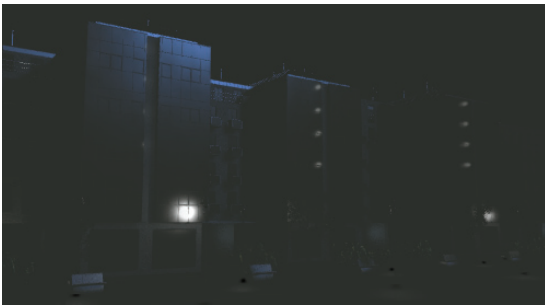
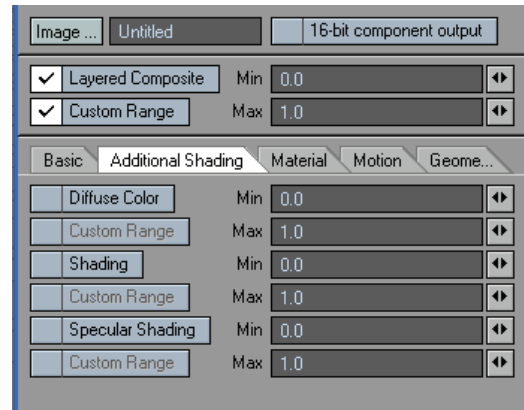


Final Render



Pre-effect Render



*Specular Color**Reflection**Diffuse Shading**Shading***Additional Shading Tab**

**Diffuse Color:** A three channel buffer, providing color options, and accounts for the part of the render for diffuse lighting, surfacing, and shadowing.

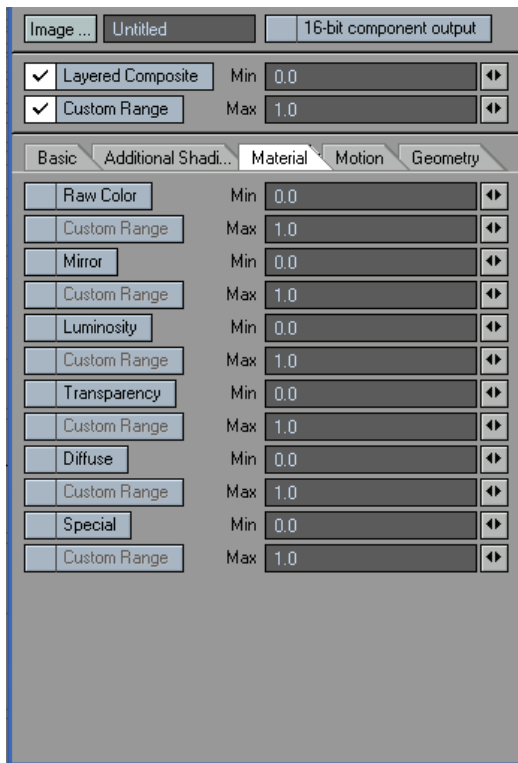
**Shadow:** A single channel buffer, indicates the location and intensity of shadows, but not shadow color. The brighter areas indicate more intense shadows.

**Specular Color:** A single channel buffer, indicates the intensity of specular highlights on a surface due to all lights, though this does not include color information.

*Diffuse Color**Shadow**Specular Shading*



## Material Tab



*Raw Color*



*Mirror*

**Raw Color:** A three channel buffer, represents the value of the Color attribute of a surface

**Mirror:** A single channel buffer, directly indicates the value of the Reflection value of a surface.

**Luminosity:** A single channel buffer, indicates the intensity of luminosity on a surface.

**Transparency:** A single channel buffer, indicates the value of the transparency attribute of a surface

**Diffuse:** A single channel buffer, indicates the value of the diffuse attribute of a surface.

**Special:** A single channel buffer, indicates the special buffer values specified for a surface in the Surface Editor. Uses only the first special buffer value.



*Transparency*



*Diffuse*



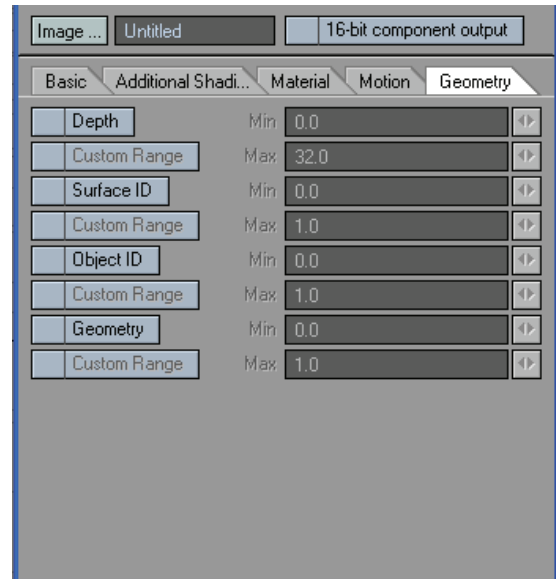
### Motion Tab



**Motion X:** A single channel buffer, indicates the direction of the geometry along the horizontal axis in relation to the camera. Negative values indicate movement towards the left, while positive values indicate movement towards the right. If the geometry is represented with a gradient, meaning parts are moving in different directions, a mid-range grey value will be used for the background.

**Motion Y:** A single channel buffer, indicates the direction of the geometry along the vertical axis in relation to the camera. A positive value indicates a downward movement, while a negative value indicates an upward movement. If the geometry is represented with a gradient, meaning parts are moving in different directions, a mid-range grey value will be used for the background.

### Geometry Tab



**Depth:** A single channel buffer, indicates the relative distance of items from the camera. The further away an item is the brighter it is represented.

**Surface ID:** A single channel buffer, indicates the internal surface identifier in Layout for each surface element in the final render, giving the ability to single-out specific surfaces in the rendered image.

**Object ID:** A single channel buffer, indicates the object identifier for each surface element in the final render. The more items in a scene, the larger the values.

**Geometry:** A single channel buffer, indicates the relation of the angle of the polygon's surface normal and the direction of the camera. The brighter the surface, the more it faces the camera directly.



Depth



Geometry

The Max field determines the maximum value allowed in a buffer. So the acceptable buffer values are always zero to the Max. All values in the buffer are divided by Max to normalize them to the range zero to one.

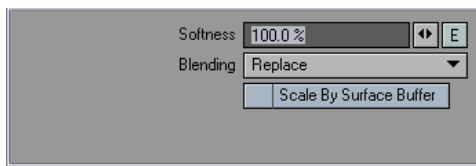


The Sliding min/max range option will dynamically compute the minimum and maximum values of each buffer. The values are computed for every frame so the output buffers will have the widest possible range of values. This is great for still images, but animations should not use this setting, due to the lack of “temporal coherence” (i.e., the images may “pop” from frame to frame).

The 16 bit component output option will save out 16 bits per channel/buffer. Normally, only eight bits are used.

## Soften Reflections *(Post-processing only)*

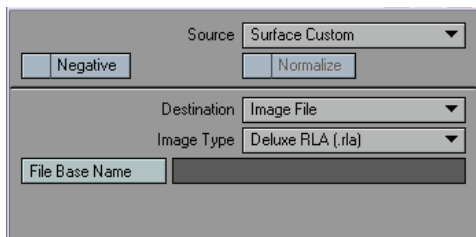
This filter will blur reflections. The **Blend** control will **Replace** the reflection with a blurred version, **Average** them together, which is more subtle, or use the **Maximum** of the replace result and the original, which avoids a dark halo, at the cost of a lighter image. You can also blend based on the **Alpha** channel or the intensity of the reflection (**LumaBlend**).



Soften Reflections can scale the blur based on the surface's value in Special Buffer 1 (**Advanced Tab** of the **Surface Editor**), if you check the **Scale By Surface Buffer** option. (A value of 1 means 100 percent.)

## Render Buffer Export *(Post-processing only)*

This filter lets you save images from one of LightWave's internal buffers (**Source**). The **Surface Custom** option on the **Source** pop-up menu will create a grayscale image, where each object surface can have a unique grayscale value. This was designed to allow post-processing effects to be applied on a surface-by-surface basis. A surface's grayscale value (0-255) is assigned using the **Special Buffer** option on the **Advanced Tab** of the **Surface Editor**.



To invert the image data, check the **Negative** option. The **Normalize** option is available only for certain **Source** selections that don't normally provide image data, like **X Motion**. **Normalize** scales the values to be 0 to 1.

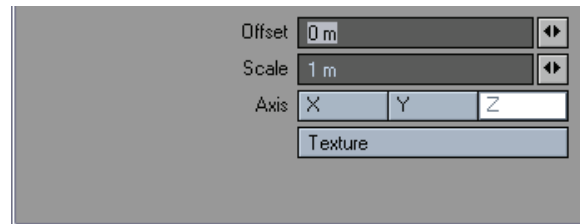
With the **Destination** pop-up menu, you can save the selected buffer image as a separate **Image File**, or replace the **Rendered RGB** or **Rendered Alpha** image data. (If you choose **Image File**, set the **Image Type** and enter the full path and filename in the **File Name** input field.)

## Textured Filter

Use Textured Filter to add the selected texture to the image before use. You could use this filter to add, say, an animated Fractal Noise pattern to a simple black image. Since textures are three-dimensional, particularly procedurals, use the **Axis** setting to use the **X**, **Y**, or **Z** of the texture.



NOTE: The differences between the **Axis** selection can be subtle.



### To see procedurals in viewports:

You can use Textured Filter to see procedural textures in your Layout viewport! Basically, you apply the procedural texture(s) to the image using **Textured Filter** and then map the image to the surface. Here's how you do it:

**Step 1:** Load an image into the **Image Editor**. It really doesn't matter what image you use since it will be obscured by the textures.

**Step 2:** On the **Image Editor's Processing Tab**, add **Textured Filter**. Double-click it in the list window to access its options. Click the **Texture** button to access the **Texture Editor**.

**Step 3:** Change the default initial **Layer Type** to **Gradient**. This provides the underlying color. You can leave it white or change it.

**Step 4:** Add one or more procedural texture layers and set as you would normally.

**Step 5:** Load your object and open the **Surface Editor**.

**Step 6:** Click the **Color** attributes **Texture** button. Leave **Layer Type** set to **Image Map** on the **Texture Editor** that appears.

**Step 7:** Set **Projection** as you would normally and select your image from the **Image** pop-up menu. The procedural will now appear in your viewport.

This operation requires a lot of computations and Layout may seem sluggish as it computes the texture.



NOTE: The image preview window on the **Image Editor** will show the texture as well. Thus, you can double-click the window to bring up the **Image Viewer**, from which you can save this compiled image. This can then be image mapped back onto the surface without all of the calculations required with **Texture Filter**.

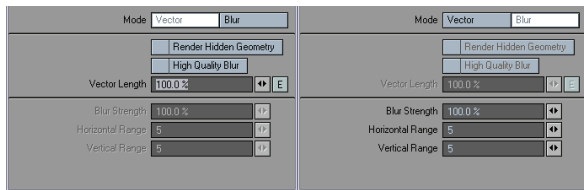
If you use an image sequence instead of a still image, you can even see an animated texture!

Note that if an animated texture is applied to a still image, it will not appear animated.



## Vector Blur (*Post-processing only*)

LightWave's normal motion blur (Camera properties) may need long render times because of multi-pass antialiasing. Vector Blur, on the other hand, can achieve great-looking motion blur in a fraction of the render time.



The two modes, **Vector** and **Blur**, use motion data that identifies each pixel's immediate motion in the horizontal and vertical directions. The **Vector** mode smears the pixel out based on the motion information, while the **Blur** mode uses the horizontal and vertical data to *blend* together the surrounding pixels with that pixel. **Blur** affects the pixels around it including backgrounds, while **Vector** alters only the pixels of the moving object.

The **Blur** mode should be used in conjunction with the normal motion blur; however, the **Vector** mode can be used by itself — you don't even need to use antialiasing! The result can be drastically reduced rendering time.

Below is a comparison between regular motion blur and the two modes of Vector Blur.



Normal Motion Blur



Left: Vector Blur: Blur, Right: Vector Blur: Vector

## Overlapping Objects

Since objects become transparent with motion blur, the filter needs something in the background of the blur. When **Compensate Background** is active, the filter does not use the backdrop and attempts to compensate for this absence. This works in most cases, but may not give a very realistic motion blur.

When **Compensate Background** is not active, the filter uses the backdrop. However, it will not show an object behind another object.

If you have overlapping objects, you may want to do some test renders to see if Vector Blur will provide acceptable results. If not, use LightWave's normal motion blur.

## High Quality Blur

If you uncheck **Compensate Background**, you can activate the **High Quality Blur** setting. This provides better quality, but takes longer to render. In this mode, you will only be able to set the **Vector Length** setting.

## Limits

The important thing to understand about using Vector Blur is that it is a *post* process. As such, hidden geometry can't be blurred and you may see problems with motion blur on shadows and moving textures. However, it can be a great help when used in conjunction with normal motion blur, by giving you better quality with lower **Antialiasing** settings (**Camera Panel**).

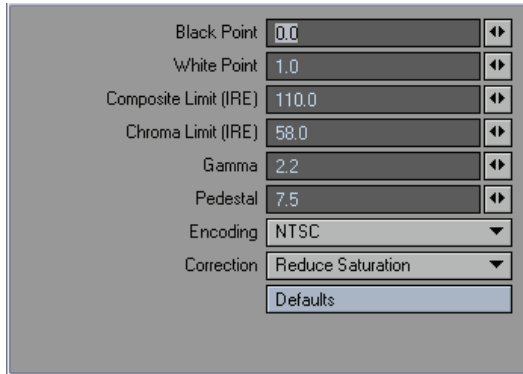




## Video Legalize

The Video Legalize filter might be more appropriately named *Hot Video*, since it assumes that pure black, RGB 0, 0, 0, is mapped to the proper *pedestal* by the encoding device (e.g., 7.5 IRE for NTSC). The encoding device may not, however, correct hot pixels — that is, colors that *exceed* certain video specifications. This is where VideoLegalize steps in.

Pixel values are generally scaled into IRE units using the **Black Point**, **White Point** and Pedestal settings as follows:  $\text{Level} = \text{Pedestal} + (100 - \text{Pedestal}) * (\text{pixel} - \text{Black}) / (\text{White} - \text{Black})$ . White is always 100 IRE and PAL Black is 0 IRE. NTSC black is 7.5 IRE, thus for NTSC, the level would be computed:  $\text{Level} = 7.5 + 92.5 * (\text{pixel} - \text{Black}) / (\text{White} - \text{Black})$ .



Normally, an RGB level of 0.0 is black and 1.0 is white. If those are used for the **Black Point** and **White Point** settings, the NTSC level will be  $7.5 + 92.5 * (\text{pixel} - 0.0) / (1.0 - 0.0)$  or just  $7.5 + 92.5 * \text{pixel}$ . When the pixel is 1.0, the level is 100 IRE, and when the pixel is 0.0, the NTSC level is 7.5 IRE.



NOTE: The actual computation is more complex than the above since other operations, like gamma correction, can happen.

The settings default to **NTSC** encoding, but you may also select **PAL** from the **Encoding** pop-up menu. (Note that you also need to click the **Default** button after selecting a new **Encoding** item.) You may change the individual settings from their defaults, if desired.

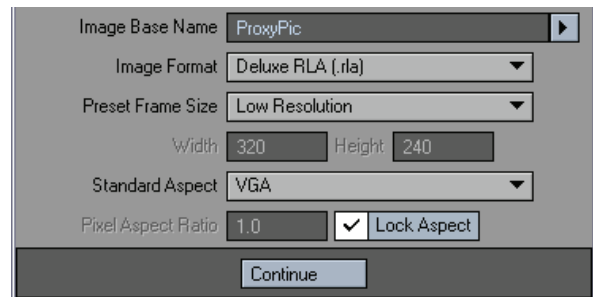
The **Correct** pop-up menu determines how the image is corrected to fall within the specified limits.



NOTE: It is highly recommended that you use VideoLegalize (as a post-process image filter) if you plan to use your images for video.

## Video Tap (Post-processing only)

The **Video Tap** filter will save a second image using different **Camera** and **Render Option** settings. This is perfect for times when you render a film resolution scene, but you want to preview a video resolution version on your TriCaster or other digital disk recorder.



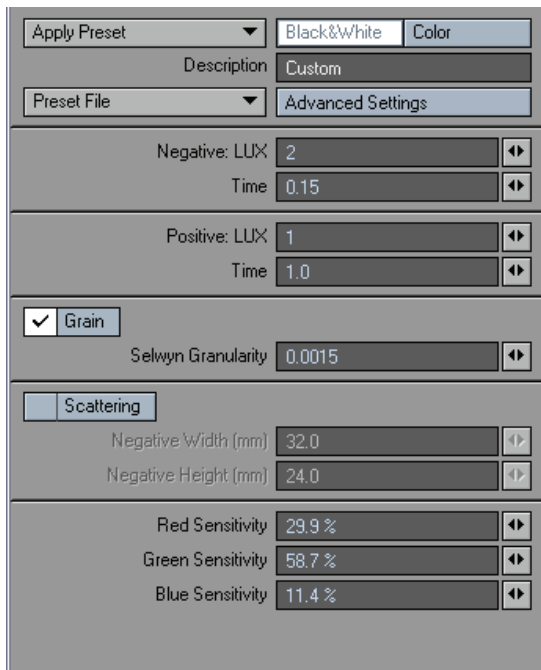
FUN FACT: Movie-makers often attach a video camera to a film camera, so they can watch dailies without developing the film—saving time and money. This filter's namesake is the **video tap on the film camera**.





## Virtual Darkroom

The Virtual Darkroom filter simulates the photographic capture of images. It is based on *A Model for Simulating the Photographic Development Process on Digital Images*, by Joe Geigel and F. Kenton Musgrave in the SIGGRAPH '97 conference proceedings.



Global settings are the four controls located at the top. **Output Type** specifies whether the output image is **Black and White** (single-plane greyscale) or **Color** (three-plane RGB).

### Basic Settings

**Basic Settings** control everything except for those settings on other tabs. **Negative LUX** is the illumination value for the negative pass — analogous to scene capture with a camera — which will affect overall brightness. **Negative Time** is the exposure time for negative pass, essentially the **Exposure** setting on the virtual camera. **Positive LUX** is the illumination value for the positive (printing) pass. Think of this as the brightness of the bulb in the enlarger or printing mechanism. **Positive Time** is the exposure time during the printing pass of virtual enlarger.

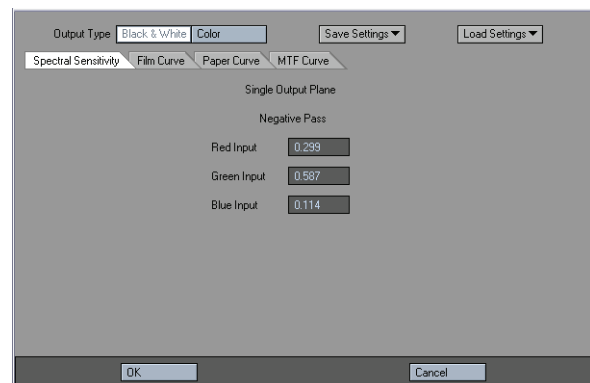
**Enable Scattering** will activate the internal scattering effect. **Negative Width** and **Negative Height** are the width and height, respectively, in millimeters, of the virtual negative. These values are used in scattering and grain calculations. **Enable Grain** will activate the grain effect. **Selwyn Granularity** controls the intensity of the grain. Increasing this value increases grain, decreasing it will decrease grain.

### Advanced Settings:

#### Spectral Sensitivity Tab

If **Output Type** is set to **Color**, there will be six sets of RGB percentage controls. Each RGB trio specifies the percentage of the black and white output plane that is contributed by the input image plane named in the control. For example, the RGB trio in the upper-middle defines how the output from the negative passes on the spectral sensitivity module creates the green plane.

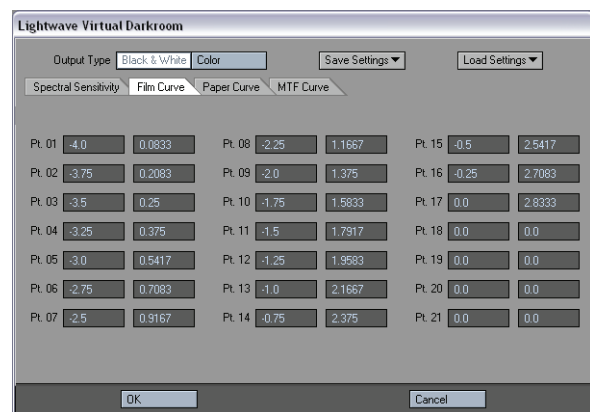
If **Output Type** is set to **Black & White**, you specify what percentage of the red, green, and blue planes of the input image are used when they are combined into a single black and white image. This transition takes place in the negative (first) pass. During the printing (second) pass, there is only a single input plane, so spectral sensitivity is not used.



*Film,*

#### *Paper, and MTF Curve Tabs*

You can enter up to 21 pairs that define the characteristic curve of the film and paper, and the function for modulation transfer used for scattering calculations. For each pair, the first value represents log (exposure) and the second value represents the density.

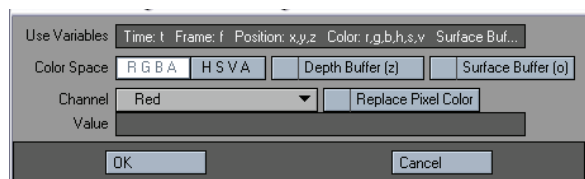


For all Curve tab entries, points should be entered in order starting with Pt. 01. If **Output Type** is set to **Color**, curves must be set for each output plane by selecting the appropriate Red Plane, Green Plane, or Blue Plane sub-tab.



## Math Filter

This pixel Filter allows you to use a mathematical formula to adjust the color in your render.



## NTSC\_Legalize

NTSC\_Legalize scans the rendered output for pixels with unsafe values of chrominance signal or composite signal amplitude when encoded into an NTSC color signal. Such illegal pixels can be corrected by reducing their **Luminance** or **Saturation**. Alternatively, the pixels can be colored black by activating the **Black Out Illegal Pixels** option. When applied, the rendered output will have a maximum composite signal amplitude of the **Max IRE** setting (110 IRE by default) and a maximum chroma amplitude of 50 IRE in compliance with the RS-170A video specification.



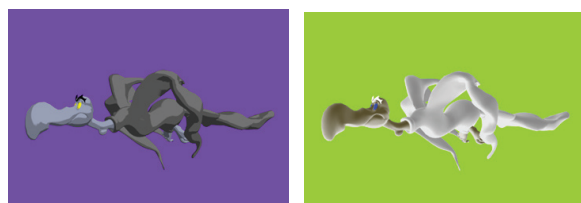
## PAL\_Legalize

PAL\_Legalize scans the rendered output for pixels with unsafe values of chrominance signal or composite signal amplitude when encoded into a PAL color signal. Such illegal pixels can be corrected by reducing their **Luminance** or **Saturation**. Alternatively, the pixels can be colored black by activating the **Black Out Illegal Pixels** option. When applied, the rendered output will have a maximum composite signal amplitude of the **Max IRE** setting (110 IRE by default) and a maximum chroma amplitude of 50 IRE.



## Negative

This plugin inverts the colors in your picture.



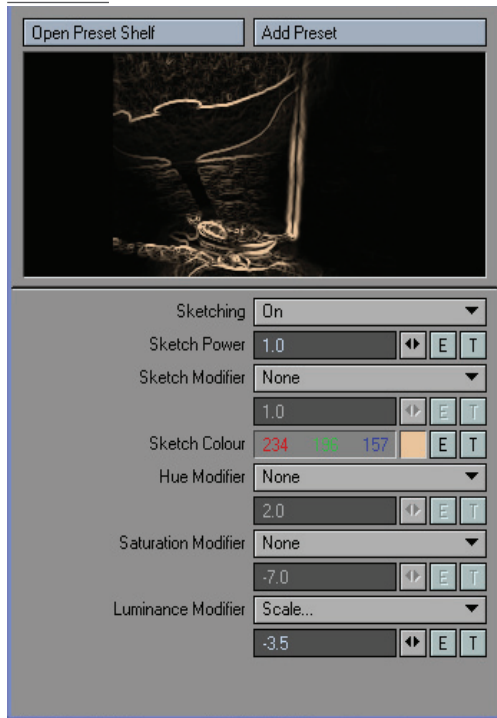
## NightVision

This darkens your image and tints it towards green giving a Night Vision-type look to your image. It also converts the picture to a low dynamic range.





## Sketch



### Controls:

Open Preset Shelf & Add Preset: Sketch settings can be added to and taken from the presets shelf. When creating a preset, the thumbnail image is taken to be the preview image currently being shown (see below).

### Preview

The preview image shows the effect of this Sketch plugin instance to the last image processed. The image will update as the Sketch settings are changed. Click on the preview image to force an update if needed.



Note that because the preview image is usually much smaller than the actual image, the preview is only an approximation of the effect.

### Sketching

Sets whether sketching is enabled or disabled. This does not affect the hue, saturation, and luminosity modifiers.

On : enable computation of sketch outlines

Off : disables computation of sketch outlines. This effectively sets the sketch value to 0.

### Sketch Power

Larger numbers makes the edges sharper. Typical values are between 0.5 and 3.

### Sketch Modifier

Modifies the “sketch” value. 0 means no edge here, 1 means a very strong edge.

None : leaves the value alone

Scale : scales the value by the given factor

Remap : arbitrarily change the sketch value, usually through an expression or texture

### Sketch Color

The color to use for the sketch edges.

### Hue Modifier

Changes the base colors of the image.

None : does nothing to the hue values

Rotate : moves the colors through the spectrum

Scale : scales the colors towards the start/end of the spectrum

Remap : arbitrarily change the hue values

### Saturation Modifier

Changes the color saturation of the image.

None : does nothing to the saturation values

Steps : quantizes the saturation values in the given number of steps

Scale : scales the saturation values

Scale around mid : scales the saturation values around the mid point

Remap : arbitrarily change the saturation values

### Luminance Modifier

Changes the color intensities of the image.

None : does nothing to the luminance values

Steps : quantizes the luminance values in the given number of steps

Scale : scales the luminance values

Scale around mid : scales the luminance values around the mid point

Remap : arbitrarily change the luminance values, usually through an expression or texture

### Gradient Parameters

For textures used as Sketch settings, a number of custom gradient parameters are available:

Sketch : the “sketch” value, between 0 (no edge here), and 1 (very strong edge here)

SketchDir : the direction of the sketch edge

Luminance : the luminance value at the current pixel

Hue : the hue value at the current pixel

Saturation : the saturation value at the current pixel

### Sketch Channels

For envelopes and expressions used as Sketch settings, a number of custom channels are available in the Sketch group. Note that these channels are read-only (they are prefixed with an “r” as a reminder), and will only contain valid values during the evaluation of the Sketch plugin.

rSketch : the “sketch” value

rSketchDir : the sketch direction

rLuminance : the luminance value at the current pixel

rHue : the hue value at the current pixel

rSaturation : the saturation value at the current pixel



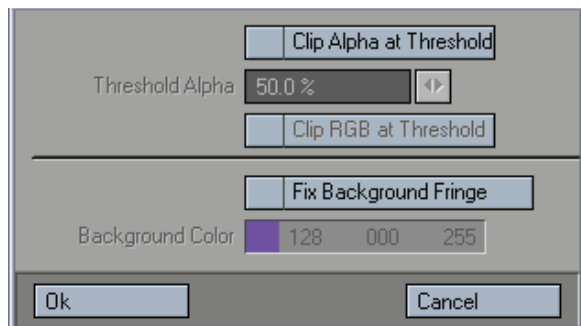
## Sepia

Adds a sepia tint to your image while desaturating it.



## SpriteEdger

The Sprite Edger image filter “unmixes” anti-aliased object edges from the background. It can remove background color fringing if the object is not rendered over black. It can also clip the alpha channel to form a 1-bit (0 or 1) mask. It can even use this mask to clip the RGB imaged edges, setting the pixels to the background color if they are outside the mask.



## VidNoise

Adds specks to a rendered image. Beware, this filter converts your render into a low dynamic range image.

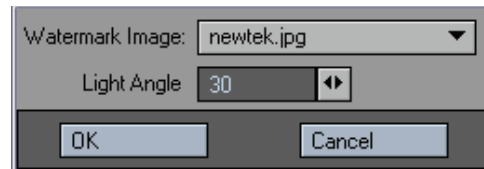
## Vignette

Creates an imaginary circle in the middle and then progressively darkens your image to the outside edge creating a central point of focus. This filter doesn't like radiosity renders.



## WaterMark

The WaterMark filter embosses an image in the lower-right quarter of your render. You can select any loaded image; however, grayscale images using a black background work best. The **Light Angle** field determines the direction of the implied light source.



NOTE: Other filters, like Pixel filters, may appear on the **Image Editor** in the list of image filters.



## Wave Filter Image

The **Wave Filter Image** filter allows you to apply and adjust image filters, color filters, grain and matte filters to the entire image, the background, objects only, shadows only, a special buffer, or a user-definable color range. A powerful interface with color preview provides the control you need to tweak your image to perfection.

Image filters include blur, sharpen, edge blend, saturation, negative, high/low limit, palette, film grain, and flip frame. The edge blend filter provides additional and faster antialiasing control, allowing a possible 30% to 50% savings on rendering times.

Color filters control RGB values, as well as the contrast, midtone, gamma, luminosity and brightness of the image. The filters work on a percentage scale, but values above 100% can be entered to create all sorts of interesting effects.

Matte filters are used to create images for compositing. Portions of your images can be set to black or white for matte creation. If you do a lot of image compositing, you will find Matte filter to be an invaluable tool.

For even more control, **Wave Filter Image** can be added more than once to allow for multiple passes over an image, with each pass applying different settings to different parts of an image and creating different effects.

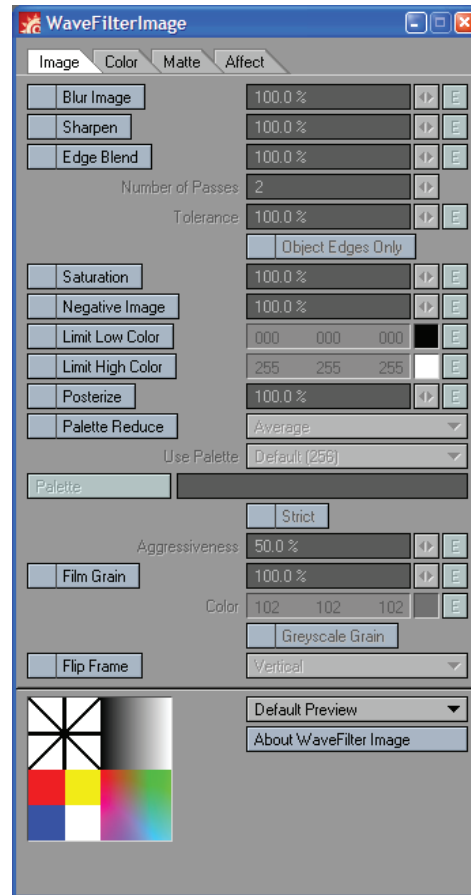
**Wave Filter Image** allows you to move many pre- and post-rendering operations directly into LightWave. It fully supports network rendering, animated settings and has batch processing capabilities.



NOTE: Special thanks to Mike Reed for his work on this plugin.

## Image Filters

Image Filters can be used individually or in combination with other filters to apply interesting effects. These filters use simple mathematical formulas for faster image rendering. Several adjustable presets are included for quick effects.



### Blur Image

**Blur Image** simulates out-of-focus effect and is applied equally across your image. This filter is commonly used on the background to help contrast your sharply-focused objects in the foreground.

### Sharpen

**Sharpen** is an edge-seeking (low-pass) detail-enhancing filter that will increase the sharpness between surface colors by exaggerating the difference in color between the two edges.

### Edge Blend

**Edge Blend** is a softening tool that will smooth the edges between sharply contrasting surfaces. This helps to tone down the stair-step edges or jaggies found in computer graphics.

You may control the **Number of Passes** this filter makes. Higher values provide finer detailed blending of the edges. **Tolerance** allows you to control the amount of edge affected. You may also limit the edge detection to **Object Edges Only**.



## Multi-Pass Edge Blend

Some images with single-pixel details, like stars or highly detailed objects, lose desired details when the Edge Blend filter is set to a strong enough value to remove unwanted jaggies. Increasing the **Number of Passes** can help to minimise this loss of detail. (Note that normally only a single pass is needed.)

When you select more than one pass, the filter searches for edges and applies the blend at a power equal to the percent you select divided by the **Number of Passes**. For example, if are using four passes at 100%, the filter will apply a blend of only 25% on each pass.

Once the first pass is completed, the edges are detected again, but this time fewer edges will be found — edges that needed only the light 25% blend will not be selected. This process is repeated for all passes. This results in the pixels that needed the least blending only getting a 25% blend, while those that needed a bit more get 50%, and so on. Only the worst jaggies get the full 100% blend.

Adding WaveFilter Image plugin multiple times can also help in removing really bad jaggies. Just like using multiple **Number of Passes**, each instance of the filter will only select the jaggies that remain after processing in any previous instance. However, remember that the blend percentage will be applied fully for each instance, so you may want to keep the value as low as possible to avoid overly blurring the edges.

## Using on Rendered images

If you render a large number of frames and find you should have used a higher **Antialiasing** setting, you can simply apply the **Edge Blend** filter to them — **WaveFilter Image** does not require geometry to work. This could save you from re-rendering the original scene.

To do this, simply load the images as an image sequence and set the sequence as the background image. Then, set the camera to exactly the same resolution as used for the original images and turn off any normal antialiasing. Set the images to render to a new filename and you're ready to post process.

## Saturation

**Saturation** allows you to control the saturation of the image's color.

## Negative

**Negative** gives control of the amount of negative applied to an image. 50% negative is equivalent to RGB 128/128/128 if the initial color is 255, 255, 255.

## Limit High/Low Color

The **Limit High Color** and **Limit Low Color** settings limit the colors in your image to levels which are PAL/NTSC compliant. Only highlights are affected.

## Posterize

The **Posterize** filter reduces the colors used causing a poster-like effect. Lower values allow more colors.

## Palette Reduce

The **Palette Reduce** filter limits the colors in your image to smaller, evenly distributed, color palettes. Palette reduction method can be specified as a preference by choosing either **Average** (the default), **NTSC/PAL Luminance**, **HDTV Luminance**, **Nearest Red**, **Nearest Green**, or **Nearest Blue**.

Several palettes are available on the **Use Palette** pop-up menu. If **Custom** is selected, the **Palette** button and input field will be available. Here you can specify a Photoshop \*.act file or an ASCII file that defines each color in the desired palette.

## Film Grain

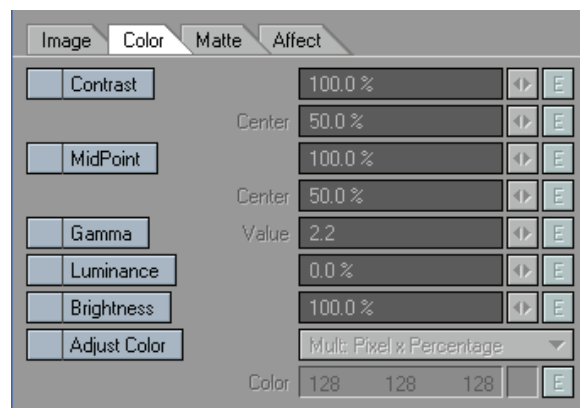
Check **Add Film Grain** to add a random noise effect of selectable density to your image. The **Color** setting defines the grain color. Checking **Grayscale Grain** causes the grain to be various levels of grey.

## Flip Frame

**Flip Frame** provides a quick method to vertically and/or horizontally flip the image.

## Color Filters

Color filters allow global control of general color attributes.



## Contrast

The **Contrast** value sets the mid-tones of the selected part of the image either towards the highlight or shadow range. The **Center** setting controls the contrast tonal range.

## MidPoint

The **MidPoint** value is similar to the **Brightness** setting, except that it affects only the mid-tones within the selected part of the image. Use in conjunction with **Brightness** for a good overall lighting effect. The **Center** setting controls the **MidPoint** tonal range.

## Gamma

Gamma allows control of the range between the darkest and lightest areas of your image.





## Luminance

**Luminance** adds overall brightness or darkness to the frame. Shadows, mid-tones, and highlights will all be affected.

## Brightness

**Brightness** multiplies the amount of light within the selected part of the image. Mid-tones and highlights are affected, while dark shadows are not affected. Use **Brightness** to adjust the lighting within your image without resetting the scene's lights or raising the ambient light level.

## Adjust Color

The **Adjust Color** options provide different options for adjusting colors.

**Mult: Pixel x Percentage** looks at the color information in each RGB channel and multiplies the base color by the selected **Color**.

**Add: Pixel + Color** adjusts the color information by increasing the amount of R, G and B color in the image.

**Sub: Pixel - Color** adjusts the color information by decreasing the amount of R, G and B color in the image.

**Replace: Pixel = Color replace** will change the RGB color from the current base color to the new selected **Color**.

## Matte Filters

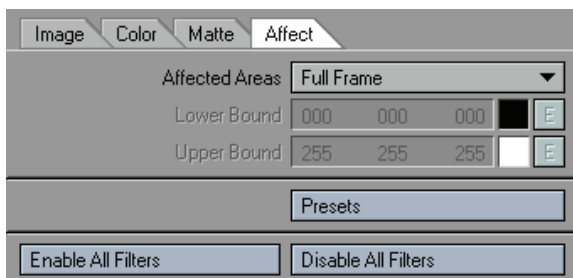


Choosing **Custom** will set the alpha channel for the affected area to a greyscale value defined in the **Custom Value** input field.

Choosing **All Black** sets the alpha channel for the affected area to black. Choosing **All White** sets the alpha channel for the affected area to white. Generally, with these, you will want **Affected Areas (Affect Tab)** set to something other than **Full Frame**.

**Invert** reverses the alpha channel for the affected area.

## The Affect Tab



## Affected Areas

WaveFilter can apply most filters to a scene's images, surfaces, channels or objects. Often, when multiple WaveFilter passes are installed, you will select a different portion of the image to affect in each pass.

**Objects** applies the effect to all objects, including an image mapped polygon background.

**Background** applies the effect to the background only.

**Shadows** applies the effect to the shadow channel only.

**Full Frame** applies the effect to the entire frame.

**High/Low Color Range** applies the effect to a selected color range in all three color channels (RGB). The range is defined by the **Lower Bound** and **Upper Bound** colors.

**Selected Surfaces** applies the effect to *selected* surfaces. Surfaces are considered selected if they have a Special Buffer value greater than zero. To access, click the **Special Buffers** button on the **Advanced Tab** of the **Surface Editor**.

## Preset Shelf

Click **Use Shelf** to access the standard Preset shelf to save and load presets. You can save **WaveFilter** settings to the shelf by double-clicking on the preview image.

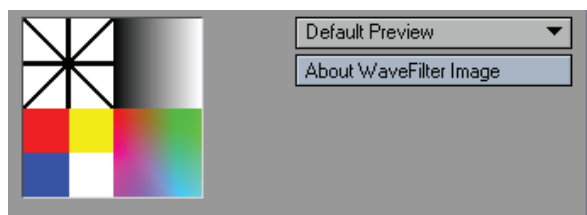
## Enable/Disable

You can quickly enable and disable all filters by clicking the **Enable All Filters** and **Disable All Filters** buttons.



## Preview Window

The preview window will give you an idea of what the current settings will do. This can take a little time to compute so you can disable the preview. You may also use the last rendered image instead of the default image.



## TimeWarp (*Camera Motion Options*)

Timewarp manipulates time. Previously time has just been a steady clock ticking away at a certain number of frames per second. Now with Timewarp you can slow the clock down, speed it up, even wind it back, all during an animation.

Timewarp can warp the time in a scene while keeping the camera unwarped. Effects like bullet-time can be created in this way. Timewarp can also manipulate motion blur in new ways for special effects. You can now fly through a motion blur.

### Timewarp basics

The Timewarp plugin is implemented as a motion modifier plugin that needs to be attached to the camera. There is also a generic layout plugin that is used to preview timewarping in layout.

It is highly recommended that fractional frames be enabled in layout when using Timewarp.

### Render frame versus Scene frame

From within the Timewarp interface, a mapping from Render Frame to Scene Frame is set through the Frame warp expression or graph.

#### Render Frame

The sequence number of a frame in a rendered animation.

#### Scene Frame

The frame number given by the frame slider.

Understanding the distinction between render frame and scene frame is crucial for effectively using Timewarp. The scene frame number is what is shown by the frame slider in LightWave. The render frame number is given by the position within a rendered animation.

Normally the render frame number and scene frame number are always the same. The 14th frame in an animation matches the 14th frame in the LightWave scene. With Timewarp however, an arbitrary mapping from render frame number to scene frame number can be established.

When rendering frames, Timewarp makes the scene frame number a function of the render frame number. By default the mapping is simply an identity function, where the render frame number is equal to the slider frame number.

As the scene frame number is now an arbitrary function of the render frame number, multiple render frame numbers may map to the

same scene frame number. While the render frame number always increases, the scene frame number may increase, stay the same, decrease, or jump around.

## Motion blur and camera warping

### Motion blur warping

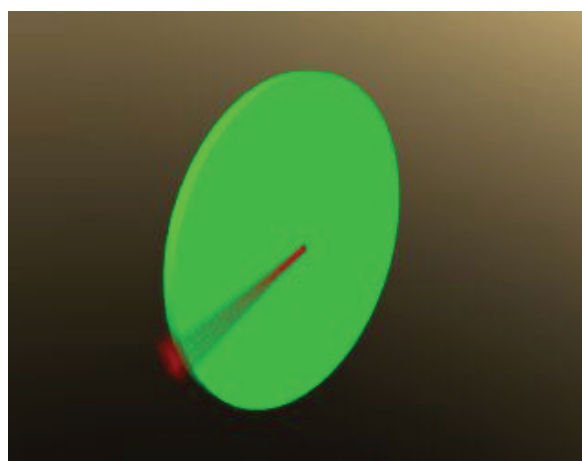
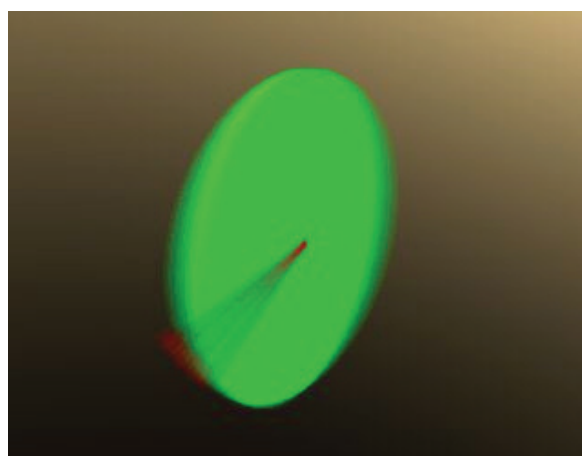
Motion blur is done by LightWave by rendering the scene at slightly different times and then combining the results. The times are computed by adding small amounts to the current frame time, up to the value given by the blur length. Timewarp can warp the blur length as well. This can keep the motion blur as it looks like without Timewarp applied, or modify it to take into account the fact that the scene is Timewarped.

### Camera warping

When mapping render frame to scene frame, Timewarp can either keep the camera as it is in the scene frame, or set it to be the same as it is at the render frame time. In the former case, the effect is similar to changing the speed of a video playback. The latter case gives effects such as bullet-time and time freeze. It disconnects the camera's timeline from the rest of the scene.

### Fixing the camera

When motion blur is enabled, the camera settings used for each pass are the settings at the motion blurred time. With Timewarp it is possible to optionally keep the camera at the unblurred time. The effect is that there is motion blur in the scene due to movement of objects, but no blur because of camera movement.





Plugin usage

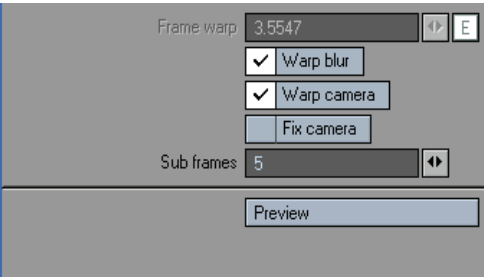
The Timewarp plugin needs to be added to the camera as a motion modifier. There can be only one instance of the Timewarp plugin in a scene.

The first step in using Timewarp is to plan your scene carefully. Plan how objects in the scene should move as a function of scene frame time (i.e. without any timewarping). If you intend to warp the camera as well, set out the camera motion as a function of scene frame time, like the other objects in the scene.

The most difficult aspect of Timewarp is if the camera is not timewarped. The camera motion is then a function of render frame time, while the rest of the scene is in scene frame time. When working with the camera, the frame set with the layout frame slider should be considered to be the render frame. When working with any other items in the scene, the layout frame slider should be interpreted as indicating the scene frame number.

To help with working with Timewarp, there is a preview mode. This presents a preview slider, which acts like the layout frame slider. When the preview is enabled, the preview slider is used to set the render frame. In layout the scene frame for that render frame will be shown. In the preview mode, it is possible to set keyframes for the camera as a function of render frame number instead of scene frame number.

Timewarp interface

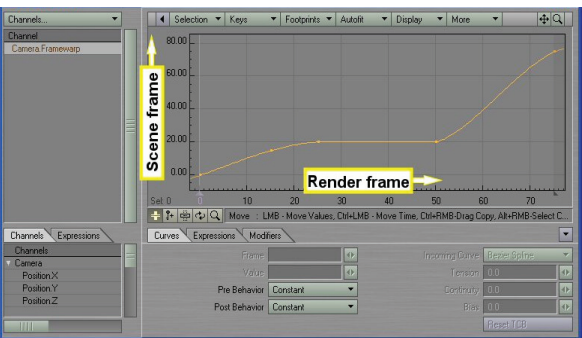


The timewarp interface is used to set the timewarp function and the various camera and motion blur warping options. The timewarp preview mode can also be launched from this interface.

Frame warp

The frame warp defines the mapping from render frame to scene frame. This uses frame numbers, rather than time. The frame numbers do not need to be integers, but can be fractional.

Although you can just set a constant here, it is more typical to use the envelope to set a graph or expression. A constant would mean that the given scene frame will be used for all render frames. When set to a graph, the horizontal axis indicates the render frame number, while the vertical axis is the scene frame number.



Warp blur

Sets whether or not to warp the motion blur. This will only have effect if motion blur has been enabled for the scene.

Warp camera

Enables and disables applying the timewarping to the camera.

Fix camera

When set, fixes the camera in place when doing motion blur. This option is not available for some combinations of options.

Warp blur	Warp camera	Fix camera available
		Yes
		Yes
		No(camera always fixed)
		Yes

If the scene does not have motion blur enabled, this option will have no effect.

Sub frames

When doing a timewarp, the computed scene frame number is very likely to be a fractional frame number. However, LightWave can only render the scene at whole frame numbers. If the fractional scene frame number were to be rounded to the nearest integer, the resulting timewarp animation will appear to stutter, especially when time is slowed down. The Timewarp plugin can overcome this by manipulating the frames per second setting for the scene.

The Sub frames setting tells Timewarp the desired sub-frame resolution. The default of 10 means that the time of the rendered frame will be accurate to within 1/10th of a frame. A value of 30 would mean an accuracy of 1/30th of a frame.

What Sub frames does is multiply the frame per second setting with the given value. For example, say the FPS is 30, and the fractional scene frame number that needs to be rendered is 1.53. Without the use of the Sub frame setting, this would be rounded to scene frame 2, an error of almost half a frame or  $0.5 / 30 = 1/60$ th of a second. With Sub frames set to 10, the FPS is modified to  $30 * 10 = 300$ , the fractional frame number becomes  $1.53 * 10 = 15.3$ , which is rounded for rendering to 15. That results in an error of only 0.3 frames or  $0.3 / 300 = 1/1000$ th of a second.

Put in another way, without Sub frames, the scene frame number would have to change by 1/30th of a second before a different frame is rendered. If the timewarp slows down time by a factor of 5, this would mean you'll only get  $30 / 5 = 6$  unique frames per second in the final animation. With Sub frames set to 10 you have up to  $300 / 5 = 60$  unique frames per second available.

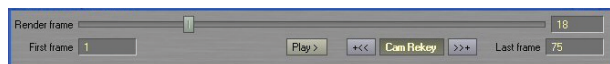
This disadvantage of setting Sub frames very large is that the number of frames in the scene becomes very large, which may have implications with memory usage. Also keep in mind that Sub frames will multiply the scene frame number. This will affect the results of any setting which is dependent on scene frame number rather than scene



frame time. For example, animated images which are animated as a function of frame number.

## Preview

The Preview button activates the Timewarp Preview interface, which is explained in the next section. Note that the preview interface can also be opened by activating the TimewarpPreview generic plugin.



## Timewarp Preview

The Timewarp Preview can be used to preview the effect of the timewarp in layout. It presents a simple slider and animation controls. The slider sets the render frame. When changed, the frame shown in layout is the scene frame that would be rendered for the given render frame number. When the Timewarp Preview is active, it is possible to set camera keyframes as a function of render frame number instead of scene frame number. This is very useful when the Warp camera option is switched off.



Note: If the camera type is anything other than Classic or Perspective, anytime-varying camera type parameters may not be correct if the camera is left unwarped. It may end up using the warped time for those parameters.

### Timewarp preview interface

The Timewarp Preview interface can be activated in two ways. The first is to use the Preview button on the Timewarp interface. The second is to start the TimewarpPreview generic plugin.

#### 6.1.1. First frame, Last frame

The first and last render frame specify the range over which the preview is done. These are initially equal to the rendering options first and last frame.

#### 6.1.2. Render frame

The slider is used to set the render frame. The view in the layout and the layout frame slider will be adjusted to show the matching scene frame.

#### 6.1.3. Play >

Toggling the Play button on starts playing the timewarped animation. The animation is looped from First frame to Last frame. Toggle the Play button off to stop the animation.

During the playing of the animation, interactions with Layout other than with the Timewarp Preview panel is blocked.

#### 6.1.4. +<< >>+

When Warp camera is off, the +<< and >>+ buttons change the render frame to the previous and next camera keyframes respectively in render frame time.

#### 6.1.5. Cam Rekey

In the case where the camera is not warped (Warp camera is turned off), the camera properties change as a function of render frame number instead of scene frame number. The layout frame slider controls the scene frame number. Normally, when a keyframe is created for the camera it will be set at the current scene frame. With Cam Rekey turned on, keyframes for the camera will be created at the

render frame number indicated by the Timewarp Preview slider.

In combination with the +<< and >>+ buttons this allows you to easily edit the camera in render frame time and the rest of the scene in scene frame time.

The Cam Rekey toggle is only available if the Warp camera option is off. Cam Rekey will only be applied when the toggle is turned on and (to avoid confusion) the Timewarp Preview interface is open.



Note: when adding a key manually using Create Key with Cam Rekey turned on, the frame number given will be ignored for the camera, and the current Render frame slider value in the Timewarp Preview used instead.



## Chapter 28: Distributed Rendering: Introduction

---



## Distributed Rendering: Introduction

### Introduction

The distributed rendering program that is packaged with LightWave is called *ScreamerNet*. It uses Layout's **Network Rendering Panel (Render > Network Render)** to control submitting scenes to networked computers running the ScreamerNet process. ScreamerNet can control up to 1,000 CPUs. Each CPU takes a frame, renders it, and then grabs the next available frame in the animation until the scene is complete.

### Are You Talking to Me?

Before ScreamerNet is discussed, your computers must be talking to each other, which means they need to be networked together. Setting up a network is beyond the scope of these instructions, though the experience can be quite rewarding.



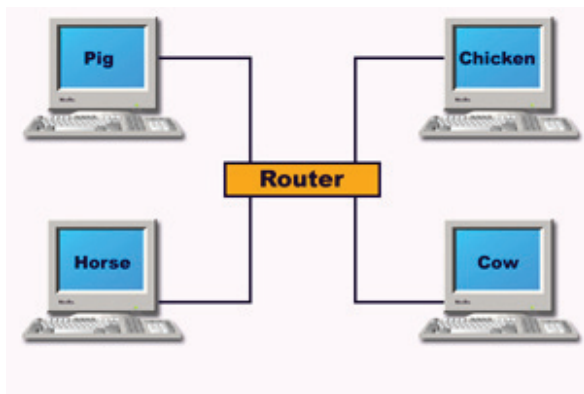
**HINT:** Since ScreamerNet communicates by writing files, NetBEUI and TCP/IP are not required. As long as each machine can see each other and write files across the network, ScreamerNet should function properly.



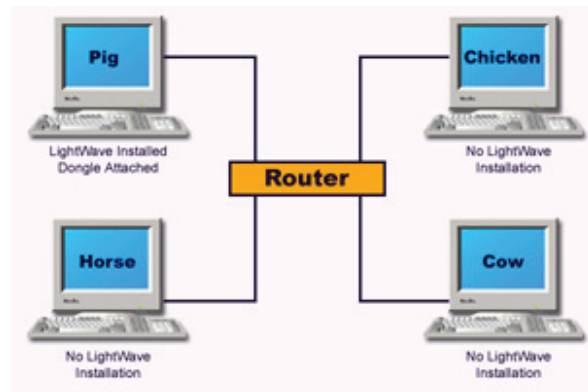
**NOTE:** If you are network-deprived, you still can use ScreamerNet to batch-render multiple scenes on a single computer. See the Batch Rendering on One Computer information near the end of this chapter.

### Share and Share Alike

Once you have created your network and the computers are talking to each other, you're ready for the next step. On our render farm, let's say that we have four computers (Pig, Horse, Cow, and Chicken) networked together. (For now, imagine that they are all single CPU machines.)



In a ScreamerNet setup, you designate one of the computers as the control or host machine. It hands out the rendering assignments, (called jobs), to the nodes (the CPUs on the network used for rendering). The host must have LightWave installed on it. In this scenario, Pig will be the control machine. Node machines do not need any NewTek software or hardware on them.



For ScreamerNet to work, all the nodes need to access specific files. Of course, all the nodes need to share the scene file and all its accompanying data including object and image files. If you're organised, all these will be tucked away in your LightWave Content directory. Also, the nodes need to save the rendered frames to a shared directory. Each node must be able to load a plugin configuration file (LWEXT9.CFG) that has the plugin files mapped with a path that the node can access. Finally, the ScreamerNet command program, (LWSN.EXE), needs to be visible to all the nodes.

Perhaps some of the confusion about setting up a ScreamerNet system arises because there is more than one way to organise a sharing scheme and you may have read tutorials that herald one method over another. If set up properly, all the schemes will work. So depending on your situation and disposition, you should choose the most suitable one for you.



**NOTE:** The following instructions refer specifically to setting up ScreamerNet on a Windows based system. You also can use ScreamerNet on Mac computers. You can find helpful tutorials for those systems on Scott Cameron's LightWave tutorial website, <http://members.shaw.ca/lightwavetutorials/>.



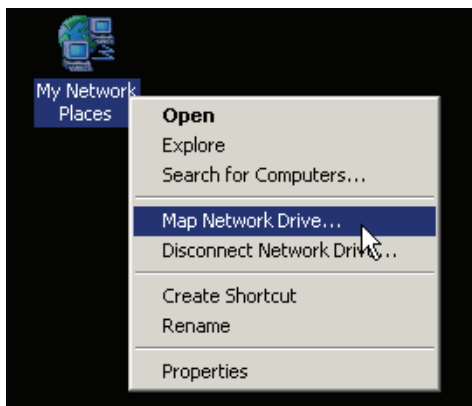


## Windows Setup

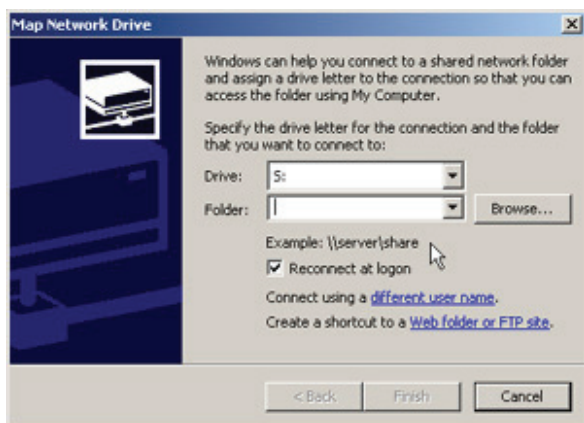
### Drive Mapping

To share, data must be at a unique address that all the nodes on the network can access. Normally, your scene file is sitting in your LightWave Content directory, which is typically on your C drive. In our case, the scene, `oldmacdonald.lws`, would be in Pig, the host computer. Now in our animal farm network, Pig, as the host, would distribute out a rendering job that said, “Render `oldmacdonald.lws` that is on the C drive.” The problem is that when Horse gets that instruction, it looks on its C drive and can’t find the scene. So Horse says, “neigh.” It cannot render a scene that it can’t find.

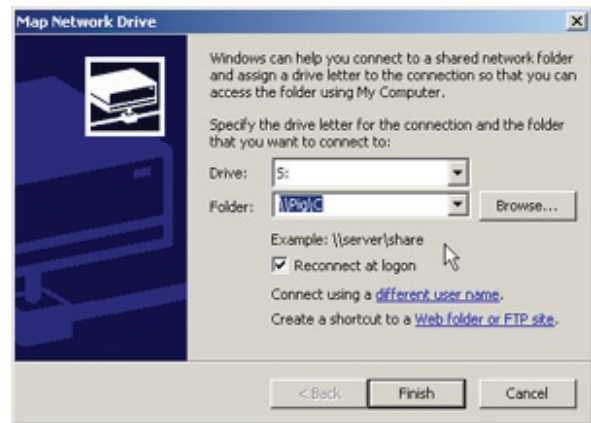
The solution is to map a unique network drive (or drives) that contains all the necessary LightWave files. Normally, these files will be on the host computer. If that is your case, go to your host computer and on your Windows screen, right click on the **Network Places** icon and choose **Map Network Drive...**



In the panel, when you specify a drive letter, make sure that it is unused by any of the computers on your network. For example, Pig might only use five drive letters, A to E, so letters F to Z are available. But Chicken may have a dozen hard drives on its computer, which means it is using drive letters A to L. If you mapped a drive on Pig with the letter G, Chicken could still be confused. In our situation, we’ll pick S for the drive letter.



Next, you specify the folder that you want to connect to this drive letter. There are two schools of thought about the folder. The orderly and efficient approach makes a folder with only the necessary files in it and attaches it to the mapped drive letter. The “kitchen sink” method simply picks the hard drive LightWave is in and selects that as the folder. For example, on Pig, LightWave is in the C drive. In the **Map Network Drive** panel, you would click **Browse...** next to the **Folder** entry box. Under Microsoft Windows Network, go to Pig and highlight the C drive. Click **OK**. The Folder box would read `\\Pig\C`.



Now if you want to be more elegant about your mapped drive, you can just map your LightWave folder. The process is the same, right click on the **Network Places** icon, choose **Map Network Drive...** and specify a unique drive letter. This time when you browse for a folder to attach, select your LightWave folder.

Both of the above methods assume that all the necessary files are in your mapped drive. If you’re organized, then all of your objects and images will be in your Content folder. Back in the old days of the 20<sup>th</sup> Century when humongous hard drives were nine gigabytes and drive space was scarce, often you would store your rendered frames on another drive. If your scene accesses any data from other drives or stores frames elsewhere, then those folders must be accessible to the network nodes. That means that you will need to map more drives.

For another way of sharing folders, Matt Gorner, (in an exhaustive, informative ScreamerNet tutorial available from the NewTek Europe website, in his interview with us.), shuns the drive mapping technique. In his method, (which you should read about directly), he simply sets up network sharing access of his LightWave folder. Matt goes into great detail about the whole process.

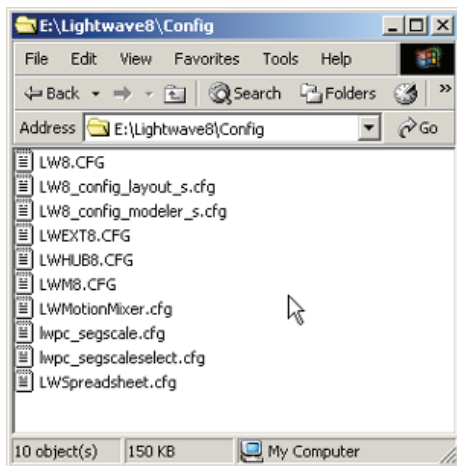
Since all the computers on your network need to share files with this mapped drive, you will need to map that drive on all your computers. After you have, you can test the connection by copying files into that drive from the node computers.



## Organising Configuration Files

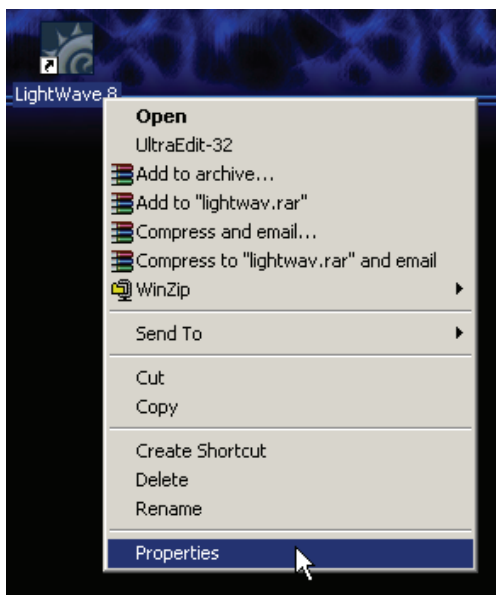
Unfortunately, we are not finished with this job. But before we move on, let's discuss alternatives for content directories. Often you will read about ScreamerNet setups that construct two file directories, one specifically for ScreamerNet and another for when you don't use ScreamerNet. This is a great deal of extra trouble to go to, given that you'll see no difference in running LightWave on your workstation under a setup for ScreamerNet, and when you decide to render across your network, you'll be ready to go without having to relocate a set of files for the scene in question. Of course, if you prefer that approach, then Matt Gorner explains a nifty way of organising a dual setup.

The next steps involve changing certain LightWave configuration files so that they are ScreamerNet friendly. Before you do, you should get those files where you can see them. Start by creating a new folder in your LightWave directory called Config. Now, click on the **My Computer** icon and do a search for **LWEXT9.cfg**, (the plugin configuration file), and move it to the Config folder. (Often it is buried in the Documents and Settings directory.) Do the same for the **LW9.cfg** file.



Next, you need to point LightWave to that Config folder when you start the program.

On your Windows desktop, right-click on the LightWave icon and select **Properties**.



Depending on your setup, in the **Target** box, it will say something like:

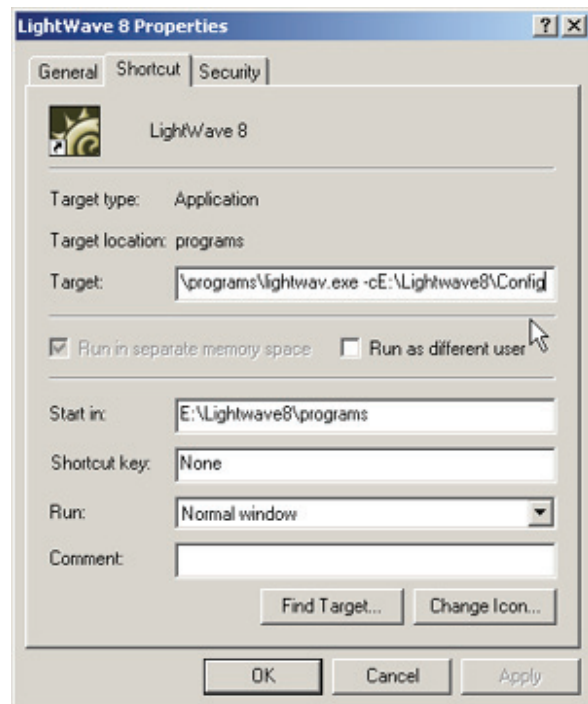
```
C:\Lightwave9\Programs\lightwav.exe
```

This points to Layout's program file.

If you have set up the Config folder, change this line to read:

```
C:\LightWave\Programs\lightwav.exe -cc:\LightWave\config
```

The first part is unchanged. The second part, ( -cc:\LightWave\config), tells LightWave where to look for the configuration files. The -c is the config cue. The rest (c:\LightWave\config) is simply the path.



**NOTE:** This is a command between you and your host computer so the path does not need to reflect the network mapped drive letters. Though if you want to remain consistent, than you would write this Target command in ScreamerNet terms. In our case, it would then read:

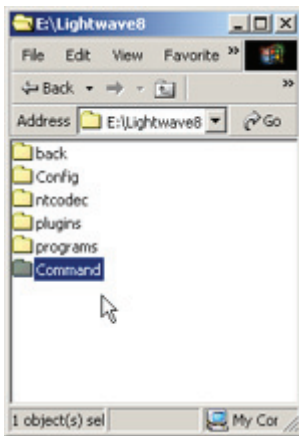
```
S:\LightWave\Programs\lightwav.exe -cS:\LightWave\config
```

Please notice that there is no space between the -c and path name.



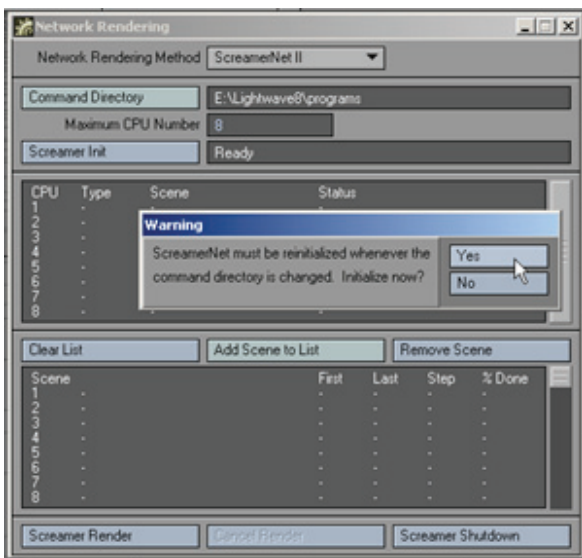
## Command Directory

When you run ScreamerNet, the host sends out Job files to the nodes and receives Ack (Acknowledgement) files back. You need to set up a Command directory to store these files. So, in your LightWave directory, create a new folder titled Command.



Now open up Layout on your host computer. Click on the **Render Tab** and under **Utilities**, click on **Network Render**. In the **Network Rendering Panel**, click on **Command Directory** and locate the newly created Command folder on your mapped drive. In our case, S:\LightWave\Command.

A message pops up that says, "ScreamerNet must be reinitialized whenever the command directory is changed. Initialize now?" Click, **Yes**. You can close the **Network Render Panel**.



Next, click **Edit > Set Content Directory**. Choose the Programs directory on your mapped drive, in our case, S:\LightWave\programs.

Finally, you need to remap your plugins. And again, there are two methods. The search and replace technique involves opening the LWEXT9.cfg file in a text editor. The lines would originally look like this:

```
{ Entry
  Class "AnimLoaderHandler"
  Name "AVI(.avi)"
  Module "C:\lightwave_9\plugins\input-output\avi.p"}
{ Entry
  Class "AnimLoaderHandler"
  Name "DirectShow(.avi)"
  Module "C:\\\\lightwave_9\\plugins\\utility\\dvview.p"}
```

You would search for C: and replace it with your mapped drive letter. In our case, it would be S:.

Save your changes and you're done. This method is great if you like to dig in there, get your hands dirty and see exactly what you are doing.

In Matt Gorner's solution, you click on the **Utilities Tab** in Layout and select **Edit Plugins**. Clear all your existing plugins and in the panel, choose **Scan Directory**. Browse to your mapped drive and highlight the Plugins folder. Hit **OK** and an Add Plugins window will pop up. Next, if the Iscripts folder is not in the plugins folder, then choose **Scan Directory** again, highlight the Iscripts folder and click **OK**. Now, if you open the LWEXT9.cfg file in a text editor, you should see that all of the plugins have been remapped to the network drive.

To save the changes to the configuration files, you need to **Quit** LightWave. When you start it up again, the new configurations will be loaded.



## Setting up ScreamerNet

All right, now that you have all the necessary files configured and in folders that can be accessed by all the nodes, it is time to set up ScreamerNet itself. In our example, we have four computers in our ScreamerNet render farm. Pig is the host and will be handing out directions to the other three rendering nodes. To keep organised, you will assign a unique number to each node.



**NOTE:** Since being the Host computer is not too taxing, you probably also will want to use it as a rendering node.

To initialise a node for ScreamerNet, you need to create a separate file that you will store and run on each node. To assist you, in the LightWave\Programs directory, you can find a file called **startlwsn\_node.bat**. Open it up in a text editor. It should read:

```
cd S:\Lightwave\Programs
LWSN.exe -2 -cS:\Lightwave\Config S:\Lightwave\
Command\job1 S:\Lightwave\Command\ack1
```

You can use this file as a guide. It consists of two lines. (Everything from LWSN.exe to ack1 is the second line.) Edit it to fit your specific setup and save it with a new name. Each computer that is a rendering node needs this file on it. You can call it whatever you want, just remember to keep the .bat extension. Be creative — YouScream!Scream.bat or SpeedKills.bat.

The first line indicates where to find the lwsn.exe program. The second line begins by running the ScreamerNet program in -2 mode. (There is a -3 mode that is explained later.) Simply, leave this part of the line as it is.

Next is a -c command, which indicates where to find the configuration files. If you have been following along, you will have set up a Config folder and moved your LW9.cfg and LWEXT9.cfg files there. If it is located elsewhere, change this path.

The next path tells ScreamerNet where to find the job commands. Again, if you have been following along, you have created a Command folder and changed the Command Directory path in Layout. The last path simply indicates where the acknowledgement (ack) files are located.

The numbers after job and ack indicate the node that is running this batch file. Each node has a unique number. Usually, if you are using the host computer as a rendering node, you assign it number 1. So, its batch file would read job1 and ack1. In our case, Pig would be node 1. You then number the other nodes. For example, Horse would be 2, Cow would be 3, and Chicken would be 4.

Each of the computers on your network needs a copy of this node initialisation batch file customised to it. For example Horse's file would read:

```
cd S:\Lightwave\Programs
LWSN.exe -2 -cS:\Lightwave\Config S:\Lightwave\
Command\job2 S:\Lightwave\Command\ack2
```

Cow's would say:

```
cd S:\Lightwave\Programs
LWSN.exe -2 -cS:\Lightwave\Config S:\Lightwave\
Command\job3 S:\Lightwave\Command\ack3
```

If any of the computers have multiprocessors, then it would need two batch files. Let's say Chicken is a dual processor machine. You would create one file for each processor. The first might read:

```
cd S:\Lightwave\Programs
LWSN.exe -2 -cS:\Lightwave\Config S:\Lightwave\
Command\job4 S:\Lightwave\Command\ack4
```

The other would say:

```
cd S:\Lightwave\Programs
LWSN.exe -2 -cS:\Lightwave\Config S:\Lightwave\
Command\job5 S:\Lightwave\Command\ack5
```

Obviously, you would have to name the batch files differently, such as StartNode4.bat and StartNode5.bat.

When you have all the batch files placed on the node computers, create a shortcut for them on the desktop of the host computer.

### Show Time

You are about ready to make the magic happen. Go back to your host computer, open Layout and load the scene that you want to render over the network. Make sure that the Content directory is set to the mapped drive.

Under the **Render Tab**, open up the **Render Options Panel**. Sometimes, you may not want to render all the frames in the scene. What you can do is set the **Render First Frame** and **Render Last Frame** fields to the desired range of frames. Make sure that you have **Auto Frame Advance** checked.

Also, check **Save RGB**. Remember, ScreamerNet only renders individual frames. It will not render animation files. In the **RGB Files** field, indicate the network-mapped path of where you are saving files. For example, S:\LightWave\Frames.

**Save** the scene.



**HINT:** You can batch render scenes in ScreamerNet. So you could save a scene and then go back and change it (for example, to render a different range of frames) and then save the scene with another name.

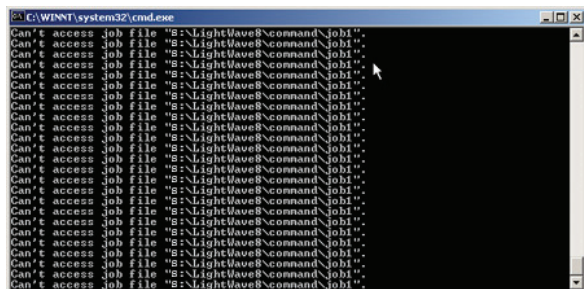
Once the scene(s) are saved, click on the **Render Tab** and **Network Render**. In the **Network Rendering Method**, choose **ScreamerNet II**, (which is probably the only choice you have).

The number that you enter in the **Maximum Number CPU** field should be the highest number node that you will be using. For example, if we would be using just node 1 (Pig) and node 2 (Horse), you would enter 2. But let's say that Horse wasn't available but Cow was. Even though, you are still using only two nodes, you would enter 3 so that ScreamerNet would look for node number 3, which is Cow.



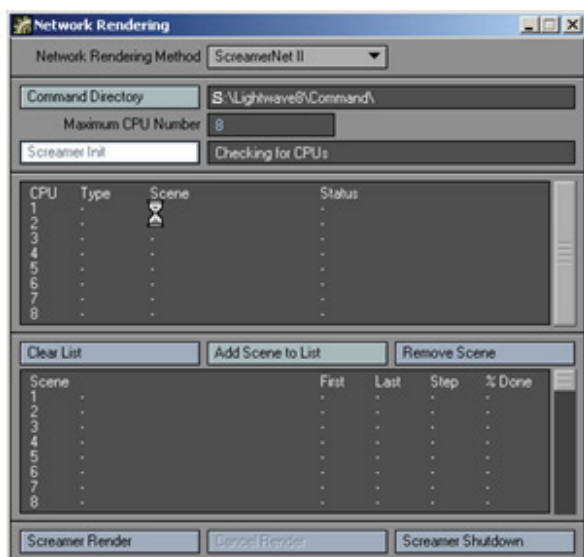


Now go to the node computers and click on the batch file icons that you created. A black MS-DOS window will open up and before you can start reading what it says, a continuous flow of repeating messages will probably begin. Depending on your exact setup, the message will read like:

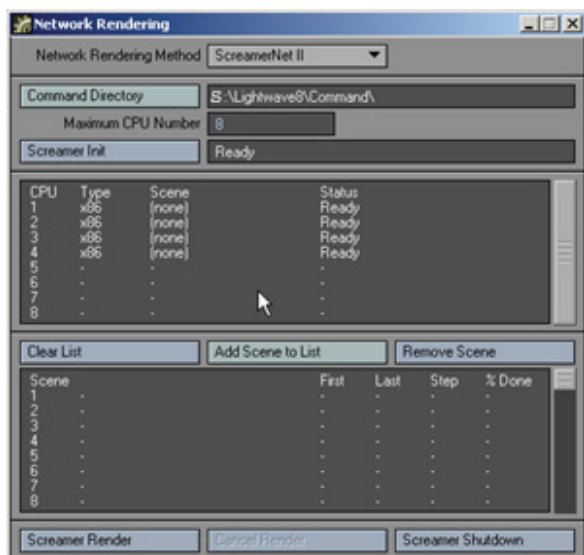


Can't open S:\Lightwave\Command\job1

This will be normal until LightWave initialises ScreamerNet. Once you have started all your nodes, return to Layout and click on the **Screamer Init** button.

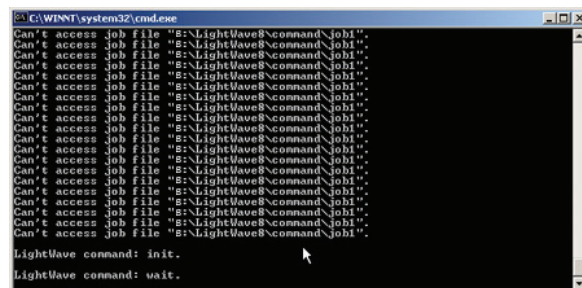


After a brief pause, you will receive a confirmation of the number of CPUs detected and the Screamer Init field will say **Ready**.



Now, if you look back at the node's MS-DOS window, the

message will say Init and then Wait. The nodes are waiting for their job commands.




In the **Network Rendering Panel**, click on **Add Scene to List** and find the scene or scenes. They will be displayed in the **Scene List** in the order they were selected. Notice that the List indicates the First and Last Frame to render, the frame advance Step, and the % of those frames that have been rendered.

If you added a scene by mistake, highlight it and click **Remove Scene**. If you want to remove all the scenes, click **Clear List**.

When the list is complete, click on **Screamer Render**. The job commands will be sent to the nodes, which are waiting in line. Node 1 receives the first frame, node 2 will take the second, and so on down the queue. If you look back at the node's window, you will see it document its progress in the render.


As soon as a node finishes a frame, it receives the job to render the next available frame in the scene. When the scene is completed, ScreamerNet jumps to the next scene in the list.



NOTE: You may wish to check some of the first frames in a photo editing program to see if they are what you expect. If you are having problems, please read the Troubleshooting section below.


When you are finished rendering all the scenes and are ready to close ScreamerNet, click the **Screamer Shutdown** button. A panel will pop up that says, "Are you sure you want to shut down the ScreamerNet CPUs?" If you choose **Yes**, the nodes are closed and their MS-DOS windows disappear. Now, if you wish to start a new session, you must restart ScreamerNet on each CPU and re-initialize the CPUs from the host machine.

If you want to stop rendering before ScreamerNet is finished, press the **Esc** key to abort the session. A message will appear in the Screamer Init window saying, "Waiting for CPUs to finish rendering." If the scene is complex or one of your nodes is slow, waiting could take a quite a while.



HINT: If you are impatient, there is no elegant way to force the nodes to quit in mid render. If you can't wait, you can close LightWave by hitting Ctrl + Alt + Del, clicking the Task Manager button, highlighting LightWave and selecting End Task. Then, you can stop each node by closing its MS-DOS window. It's brutal, but it works.



 **WARNING:** If you close the MS-DOS window on any of the render nodes, you may have to restart all nodes and re-initialize them.



## Batch Rendering on One Computer

Even if you only have one computer, ScreamerNet can be useful for rendering a series of scene files unattended. You simply set your computer as both the host and a node. You don't need to network map your drive, but you will need to create a batch file to start the node. The rest of the procedure is the same. Of course, if you are using a dual-processor machine, each processor should be treated as a separate CPU for rendering.

## Rendering Without LightWave

The LWSN program has a third option that lets you render a scene without running LightWave. There is no control machine and thus it is not a distributed rendering situation. You must tell the program specifically what to render. The method is run from a DOS prompt using the following syntax (one line):

```
LWSN -3 [-c<config dir>] [-d<content dir>] <scene file> <first frame> <last frame> [<frame step>]
```

As you can see, you supply the program with the basic information needed to render a scene. An example would be:

```
LWSN -3 -cS:\Lightwave\Config -dS:\Lightwave\Content oldmacdonald.lws 1 900 1
```

In the example, the program would render frames 1 through 900 of the oldmacdonald.lws scene using the config files stored in the S:\Lightwave\Config and using S:\Lightwave\Content as the Content Directory.



HINT: You can get the syntax for ScreamerNet by simply typing LWSN with no arguments.

## Troubleshooting and Limitations

The most common cause of ScreamerNet crashing is when too many nodes try to write or read their information to or from the host computer while the host renders. If that seems to be the case, try saving the frames to a mapped network drive on another computer. You will have to go back to your scene and redirect the path for saving frames. Of course, make sure that the new drive has enough space to store the frames.

If that doesn't solve your crashing problems, do not use the host machine as a render node. Use it only as a server where the hard drives are found.

ScreamerNet can be finicky about how filenames and directories are composed. Putting spaces in a name, for example, old macdonald.lws, might cause a problem. Try using a dash, -, or an underscore, \_, instead of spaces, old\_macdonald.lws. You should also never use accented characters such as é, à or ç since there is no guarantee that these characters will all be rendered correctly through your farm.

Some plugins don't like running with more than one thread. For ScreamerNet scenes, go to the **Render Options Panel**, (**Render > Options > Render Options**). In the **Multithreading** drop down menu, change to **1Thread**, and save the scene. **Quit** LightWave and start it again. This will update the configuration files. You then will need to restart all the nodes so they will read the updated configuration files.

Hopefully everything is running smoothly, but problems can occur, usually because something is setup wrong. Some of the main problems that arise are listed here along with possible remedies.

The biggest limitation to ScreamerNet are plugins that are not written to take advantage of it. I'm not aware of a definitive list of all the ones which do and don't work. If in doubt read the documentation that came with the plugin, it should say whether it has problems or not. If it doesn't say but the plugin still doesn't seem to work, re-scan your plugins to update the LWEXT9.cfg file. If it still doesn't work then chances are it's not compatible with ScreamerNet.

If your scene uses procedural textures (fractal noise etc.) you may experience differences in the pattern if you render on machines with different processors. Mixing old and new processors; AMD and Intel; could lead to problems. This is really only a problem with animations as textures could suddenly change from frame to frame, if in doubt run a test or use all computers with the same processor type.

Another known problem with ScreamerNet (correct as of v7.5c of LightWave) is its dislike for scenes that have had a Spreadsheet used on them. Plugins like Spreadsheet save data to the scene file, which causes ScreamerNet to hang. The only way around this is to remove the entry in the scene file made by Spreadsheet.

Open your scene files in a text editor. Near the top you will find an entry that says:

```
Plugin MasterHandler 1 .SpreadsheetStandardBanks
```

followed by:

```
EndPlugin
```

There will be another right after that entry that starts with:

```
Plugin MasterHandler 2 SpreadsheetSceneManager
```

and if you scroll down there will be another:

```
EndPlugin
```

Highlight all the text between the first Plugin MasterHandler and the last EndPlugin, delete it, then save the file, it should now render properly.

### Problem:

"My nodes can't find the job files."

### Possible solution:

They can't find the job files because they can't see the Command folder, which holds them. So open up the node batch file and check the script lines that end in job and ack. Make sure that pathname before these words point to your Command folder.

Also check that all the render node computers have access to read and write to the folder.



**Problem:**

"My nodes seem fine but when I press Screamer Init it can't find any CPUs."

**Possible solution:**

Again this is a command folder problem, check that the **Network Render Panel** has the same Command Directory path as the nodes batch file. Also check that the host computer has access to read and write to the folder.

**Problem:**

"Rendering seems to be working but no files are saved."

**Possible solution:**

The messages output in the node's MS-DOS window will often give you a clue to the problem. For example, it might say that it could not find an object or image that was in the scene. This can be caused when the Content directory that the rendering node is accessing does not contain all the scene's data. Check that the render batch files are looking in the correct place for the config files, and that they can access the folder on the network.

Next, make sure your scene file is saving **RGB FILES** and **not** an animation (.MOV / .AVI etc.) in the **Output Tab** of the **Render Options Panel**. ScreamerNet can **ONLY** save individual frames.

Finally, check that your plugins (LWEXT9.cfg) file is up to date by re-scanning them using the pathname all the nodes will be using to find them on the network.

**Problem:**

"Rendering seems to work okay but all my saved files are in .FLX format."

**Possible solution:**

There are two possible reasons. The first is that ScreamerNet can't find the plugin to save in the format you've specified, so check that your plugins (LWEXT9.cfg) file is up to date by re-scanning them using the pathname all the nodes will be using to find them on the network. Also check that the render batch files are looking in the correct place for the config files, and that they can access the folder on the network.

The second reason is that the render node that saved the file ran out of memory to load the saver plugin so, as a last resort, it used the built-in FLX saver.

If you have a whole scene's worth of FLX images, you can use LightWave to convert them into your desired format. Simply load the series as a sequence of images for the **Backdrop** in a blank scene and render in the proper format. If you only need to convert a few FLX images, load them into the **Image Editor Panel**. Highlight the name of an image and double-click on the preview, (or just double-click on the name), to open it up in the **Image Viewer**. Now, click on **File > Save RGBA**, and pick your format.

**Problem:**

"My particles aren't working."

**Possible solution:**

Particles are a mathematic simulation. Therefore, each computer would start calculating the particles at the start of each frame rendered. The way to fix this is to "bake" the motion of the particles. You need to save the particle FX calculation to disk (as a .PFX file) in the folder with the scene file, or to a mapped drive that can be seen by all the networked computers.

**Problem:**

"Motion designer isn't working."

**Possible solution:**

Same as particles, this is a mathematic issue. The way to handle this is to bake the motion, and save the motion designer calculation to disk (as a .MDD file) in the folder with the scene file.



## Mac OS X Setup

### Foreword

LightWave's LWSN (LightWave ScreamerNet) allows standalone, batch and network rendering across multiple platforms including Mac OS X.

Each section builds upon the knowledge learned in the previous sections, so for best results you should work your way through this entire tutorial from the beginning, rather than skipping right ahead to the network rendering sections.

LightWave's Config Files are plain text files that store basic configuration information for LightWave and LWSN. It's very important that you know where these config files are located so that both LightWave and LWSN can access them. It's also important to understand which LWSN relevant settings are stored in the config file, and how to use them.

### Updating LightWave's Config Files

LightWave only updates the config file when you quit the program. Therefore anytime you make changes to any of the config settings you must be sure to quit LightWave to ensure that the changes are written to the config file itself. LWSN only reads the config file when first launched. Therefore, if you wish to make changes to the config file for LWSN and you are using a single config file for both Lightwave and LWSN, you should perform the following steps:

1. Quit LWSN.
2. Launch LightWave.
3. Make changes to the desired config settings in LightWave, as discussed later.
4. Quit LightWave to write the changes out to the config file.
5. Re launch LWSN so it reads the updated config file.
6. Re launch LightWave if you are using the built-in network controller to manage LWSN.

## LightWave's Config File Path on Mac OS X

On Mac OS X the default location of the config files is:

"YourHD:Users:username:Library:Preferences"

Replace YourHD with the name of your hard drive where your home directory is located and replace username with your user name.



NOTE: Specify all paths for LightWave using the Macintosh path separator ":" (colon), not the Unix separator "/" (forward slash) nor the DOS separator "\" (back slash).

Keep in mind that, by default, the config files are stored in the Library:Preferences folder that's inside your home folder, not in the Library:Preferences folder at the top level your system hard drive.

The config files for LightWave 10 on Mac OS X are named as follows:

LightWave Extensions 10 Prefs : Complete list of file paths to all available plugins

LightWave Hub 10 Prefs : Basic Hub Settings

LightWave Layout 10 Prefs : Basic Layout Settings

LightWave Modeler 10 Prefs : Basic Modeler Settings

You don't need to specify any of these config files by name, only the path to the directory where they are stored, and that's only if you decide to keep your config files in a location other than the default. If you leave the config files in their default location, you do not need to specify their path for LightWave, Modeler, the Hub, or for LWSN. In this case each individual Mac OS X user would also have their own configs, since they would be stored in each user's home folder.

You may wish to store your configs in different locations in some advanced situations, such as:

- When running different versions of LightWave on the same machine.
- When different users use the same Mac and you all wish to use a single set of configs.
- When running ScreamerNet over a network with a common set of configs for all nodes.
- When running ScreamerNet over a network and using separate configs for machines with varying resources, such as available RAM or number of processors.



## Creating a New Set of Config Files

The following steps will show you how you can easily create a fresh new set of config files in a new location if desired.

1. Locate your LightWave installation, typically:  
YourHD:Applications:LightWave10
2. Inside the LightWave10 directory, create a new directory named Configs.
3. Locate the LightWave cmdLine file, typically installed in: "YourHD:Applications:LightWave10:Programs"
4. Double click it, to open it in TextEdit.
5. Type the config parameter -c immediately followed by the full path to the new Configs directory, enclosed in quotes, as follows:  
-c"YourHD:Applications:LightWave10:Configs"



NOTE: Do not put a space between the config parameter -c and the config path. Also make sure you type the parameter -c using a lowercase c.

6. Open TextEdit->Preferences, Saving and turn OFF: Append ".txt" to plain text files. If there is a ".txt" extension added to the end of the LightWave cmdLine file name, it unfortunately won't be recognized by LightWave.
7. Save the LightWave cmdLine file.
8. Copy and paste the config parameter, -c"YourHD:Applications:LightWave10:Configs", into the Modeler cmdLine file and the Hub cmdLine file and save each of those files as well.
9. Launch LightWave.
10. Open the Edit Plugins panel with Utilities->Plugins->Edit Plugins...
11. Click Scan Directory, & Choose: :Applications:LightWave10:Plugins
12. Close the Edit Plugins panel by clicking Done.
13. Quit LightWave, which will write out a new set of config files in the new Configs directory.
14. Launch & Quit Modeler so that Modeler creates its config file.
15. Quit the Hub so that it creates its config file.

If you run into various problems with LightWave and suspect corrupted config files, you can delete them and use the above steps #9-15 to recreate a fresh new set of config files in their current location.

## Config Settings for LWSN

The following settings are stored in the LightWave Config file named: LightWave Layout 10 Prefs and are used by LWSN: Content Directory, Command Directory, Default Segment Memory, and Multithreading.

## Content Directory

ContentDirectory YourHD:Applications:LightWave10:Content:

Along with a full compliment of paths to the Content Directory's special subdirectories, such as :Images, :Objects, :Scenes, etc. If you wish to use LWSN successfully you must learn to properly use LightWave's Content Directory structure to package your projects correctly.

A Content Directory in LightWave is very similar to a Web site's root directory. When used properly, all necessary file paths are specified relative to the root directory, and all files are located in subdirectories of the root directory. This means the root directory is a self contained unit that may be moved as desired, including to other machines, without breaking all the dependent file/directory links.

In LightWave, the Content Directory itself may be named whatever you like, but there are some specific subdirectories that LightWave expects to find inside the Content Directory that you should not rename or move. These special subdirectories are named Images, Scenes & Objects. They are where you should store all your image, scene and object files respectively, for a project. You may name your files anything you wish (being sure to use proper file extensions), as long as you place them in the proper subdirectories.

Here's an example of a very simple Content Directory:

The ContentDirectory entry is not actually written to the config file unless you specifically set a Content Directory in LightWave as follows.

1. Launch LightWave
2. Select: Edit->Set Content Directory...& Choose your desired Content Directory.
3. Quit LightWave, to save the config file.



## Command Directory

CommandDirectory YourHD:Applications:LightWave10:Programs:

The Command Directory is the directory that contains the job command file and the acknowledgement file. These are two plain text files that the network controller uses to communicate with an instance of LWSN during batch or network rendering.

The communication goes like this:

- 1.The network controller creates a job file.
- 2.The network controller writes a command into the job file.
- 3.LWSN reads the command from the job file.
- 4.LWSN attempts to perform the command.
- 5.LWSN creates an acknowledgment file.
- 6.LWSN writes a reply to the acknowledgement file.
- 7.The controller reads the reply from the acknowledgement file to decide what's next.

These files are named: job# and ack# where # is replaced with the number of the LWSN node to control. If only one instance of LWSN is being used these files would be named job1 and ack1. For a second instance of LWSN, they would be named job2 and ack2, etc. As in config and cmdLine files, no ".txt" extension is used. Also note that there is no leading zero, space or anything else between the number and the words job or ack.

LightWave defaults the Command Directory to the LightWave Programs Directory. LWSN on Mac OS X however uses the Content directory as the default Command Directory. This means it will not run as configured with the defaults, you must at least change one of these two settings.

Both the user running the network controller (typically LightWave or a third party controller) and the user running LWSN (if different) must have read/write access to the same Command Directory. Non-administrator users don't typically have write access to the Applications directories on Mac OS X. Therefore, it is suggested to use a shared Content Directory, with read/write access for all users, and keep everything relative to this shared Content Directory, including my Command Directory.

A typical Content Directory, with their own Commands directory may look like this:

Here's a step-by-step example of how to configure such a simple self contained Content Directory with its own Command Directory.

- 1.Launch LightWave.
- 2.Set LightWave's Content Directory with: Edit->Set Content Directory to your desired Content directory.
- 3.Select: Render->Network Render
- 4.Click the Command Directory button.
- 5.Navigate to the same Content Directory you set in step 2.
6. Create a New Folder named Commands inside the Content Directory.
7. Click the Choose button to close the dialog and accept the changes.
8. If asked to initialize the ScreamerNet, click No.
9. Quit LightWave to save your changes to the config file.
10. When creating the LWSN cmdLine files (fully explained later) use the following Content relative command file paths, replacing # with the number of the LWSN node:

“:Commands:job#” “:Commands:ack#”

## Default Segment Memory

DefaultSegmentMemory 32000000

This is the number of bytes to use for the rendered image segment memory. It defaults to 32 million bytes which is just under 32 Megabytes. This only affects the memory allocated for image buffers used in rendering the image itself, it doesn't affect the amount of RAM used to load objects, textures etc. The number you set is the maximum upper limit. LightWave will only use as much RAM as it needs for the actual render buffers, up to this limit. If more memory is needed, LightWave will break the image up into as many segments as necessary so that each fits within the segment memory limit. In general you want this number set high enough for an entire frame to be rendered within one segment, as long as it fits comfortably within your machine's physical RAM capacity, while still leaving enough physical RAM free for objects, textures etc.

This value is set in the Camera Properties panel (typically while a scene is open) but is stored in the Config file, only if you say yes when asked to use the changed value as a new default. It is never stored in the scene file, and is not stored anywhere if you click no when asked to use the changed value as a new default. In that case it will only affect a render in LightWave itself until you quit LightWave. The next time you launch LightWave the old value in the config file will be reinstated.

Here's a simple step-by-step example of how to set the Default Segment Memory

1. Launch LightWave.
2. Open the Camera Properties panel.
3. Set the resolution to whatever you wish.
4. Notice the readout directly under the Segment Memory Limit button.
5. If it reads Segments:1 you don't need to change it, otherwise continue on...
6. Click the Segment Memory Limit button.
7. The Segment Memory Limit dialog will open, filled with the current DefaultSegmentMemory value from the config file.



**FUN FACT:** This number is in millions of bytes, not megabytes. The rounded integer shown is simply multiplied by 1,000,000 when stored in the config file. A true Megabyte of RAM is actually  $1024 * 1024 = 1,048,576$  bytes.



8. Enter an integer that is equal to the old number multiplied by the number X that was displayed in the readout Segments:X from step 5. For instance, if the old value was 32, and the readout said: Segments:2, you could enter 64 into the field, to be sure to be able to render the entire frame in one segment.

9. Click the OK button.

10. You will now see a dialog that asks: "Should this value become the new default?" Yes/No. You must click the Yes button, or the new value will not be stored anywhere and will only persist during this session of LightWave until you quit LightWave.

11. The readout below the Segment Memory Limit button should now read: Segments:1

12. Quit LightWave to actually save the change to the config file.

You may also simply locate and edit the line DefaultSegmentMemory 32000000 in your config file directly in a text editor. Simply change the leading integer from 32 to whatever you desire (followed by six zeros) and save the config file. Make sure your text editor keeps the file as a plain text file and doesn't add any file extension to the config's filename.

## Multithreading

### RenderThreads 1

This is the number of concurrent threads (strings of commands) to execute during rendering. This setting is most often used on multiprocessor machines, but in some instances may even speed up single processor rendering. It's best to run various tests on your particular scenes and machines to determine the best possible settings in your case. When setting render threads higher than 1, be sure to check test renders for any problems. Some plugins may not work properly with multiple threads, and may crash.

In general when using such a dual processor machine in a render farm, you'll get the best results running two instances of LWSN, each using one thread. One thing to keep in mind though is that running two instances of LWSN means you'll be using twice as much memory as a single instance. So if memory is an issue, such as when rendering high resolution images, you may actually get better results running one instance of LWSN set to two or more threads. That way both CPUs and all available memory can be used for a single image. Luckily you can stuff the dual processor Power Macintosh G5's with up to 8GB of RAM which can enable you to render massive scenes.

Since the Multithreading setting is stored in the config file, and LightWave & LWSN run optimally with different render thread settings, it is often useful to configure LightWave and your LWSN nodes to use different config files. Otherwise, be sure to check/set the RenderThreads setting in LightWave (or directly in the config file with a text editor) prior to launching LWSN.

Here's a step-by-step example of how to set the RenderThreads setting with LightWave:

1. Launch LightWave.
2. Open Render->Options->Render Options...
3. Click on the Rendering tab.
4. Set the Multithreading pop-up menu to 1, 2, 4 or 8 Threads, as desired.
5. Quit LightWave, to have LightWave write the change to the config file.

You may also simply locate and edit the line RenderThreads 1 in your config file directly in a text editor. Simply change the number to 1, 2, 4, or 8 as desired and save the config file. Make sure your text editor keeps the file as a plain text file and doesn't add any file extension to the config's filename, or it will not load into LightWave.

LightWave uses command line parameters which in Windows are specified in a DOS window or a shortcut icon. NewTek has added a simple mechanism for specifying command line parameters on the Macintosh as well. Plain text files called cmdLine files are used to store any command line parameters that you wish to pass to a LightWave application when it is launched. LightWave, Modeler, Hub and LWSN can all use their own cmdLine files.





## cmdLine File Names

cmdLine files are plain text files that are named exactly the same as their associated application (which must be one word), followed by a single space and the word "cmdLine". They do not (and can not) have a file extension added (not even ".txt"), or they won't work. The format is as follows:

SingleWordAppName SPACE cmdLine NO EXTENSION

The default names of major applications and cmdLine files (not all Programs files are shown) in the LightWave10:Programs folder are as seen in the example below left:

When setting up multiple instances of LWSN to use in batch or network rendering, you'd make multiple copies of the LWSN application and its associated cmdLine file and rename each pair to match, typically using an incrementing number as in the above right example.



Note: The icons for the cmdLine files may change to text document icons once you edit them.

These cmdLine files must be located in the same directory as the applications that they are associated with. Typically: "YourHD:Applications:LightWave10:Programs" where YourHD is replaced with the name of your hard drive that holds your applications.

## General cmdLine Syntax

The command line syntax for the contents of a cmdLine file is as follows, all on one line with a space separating each parameter. Do not separate a parameter from its value with a space. All parameter letters must be in lowercase. Enclose all paths with double quotes, especially if the path contains spaces.

[ -0 ] [ -c<config directory> ] [ -p<plugin config dir> ]

The following general command line parameters may be used in most of the cmdLine files:

### Hub Switch

-0 : (dash zero) Disables the Hub

This parameter may be included in the LightWave and/or Modeler cmdLine files if you wish to disable the hub. If you are experiencing problems with LightWave and/or Modeler, try disabling the hub, which is often a source of problems on the Mac. Otherwise, launching either LightWave or Modeler will also launch the Hub.

## Command Line Path Parameters

When specifying command line path parameters, the following rules of thumb will help avoid problems:

- Type the parameter, -c or -p, letter in lowercase.
- Do not put a space between the parameter letter and the path.
- Enclose all paths in double quotes, especially if the path contains any spaces.
- Avoid using spaces in file or folder names inside the content directory, use underscore.
- Prefix all relative paths with a leading colon. Such relative paths are relative to the Content Directory. For example: " :Scenes:MyScene.lws"

## Config Path

-c : Optional: Path to the directory that contains the config file. The default is as follows:

-c "YourHD:Users:username:Library:Preferences"

If not specified, the config directory defaults to the current user's preferences directory, which is normally as specified above, where YourHD should be replaced with the name of your hard drive that holds your home directory and username should be replaced with your user name. Therefore, if you leave your config files in their default location in your user preferences directory, you do not need to specify this option in your cmdLine files and each individual user could have their own configs. This parameter may be used in LightWave, Modeler, Hub and/or LWSN cmdLine files to specify a different directory for the config files.

## Plugin Database Path

-p : Optional: Path to the directory that contains the plugin database file. Normally as follows:

-p "YourHD:Users:username:Library:Preferences"

If not specified, the config directory will be used. This parameter may be used in LightWave, Modeler, Hub and/or LWSN cmdLine files to specify a different directory for the plugin database file: LightWave Extensions 3 Prefs

## General LWSN cmdLine Syntax

There are a few common command line parameters that are used to tell LWSN what to do. The following command line parameters may be used when running LWSN in any mode. Additional mode specific command line parameters will be discussed later.

You tell LWSN which mode to use, as well as what to render through the use of the LWSN cmdLine file. The general common LWSN cmdLine syntax is as follows, all on one line.

-<mode#> [ -c<config directory> ] [ -d<content directory> ] [ -q ]





## Rendering Mode

-<mode#> : The mode number parameter specifies which rendering mode to use.

-2 : Signifies batch or network rendering mode which may render multiple scenes on multiple nodes.

-3 : Signifies standalone rendering mode which renders a single scene on a single node.

## Config Path

-c : Optional: Path to the directory that contains the config file. Default is as follows:

-c "YourHD:Users:username:Library:Preferences"

This parameter is explained in more detail above under Command Line Path Parameters.

## Content Path

-d : Optional: Path to the directory that contains your content. For example as follows:

-d "YourHD:Users:username:Documents:LWContent"

Where YourHD should be replaced with the name of your hard drive that holds your home directory and username should be replaced with your user name. This example would be to use a directory named LWContent in your Documents directory as your content directory. If this option is not specified, the content directory defaults to the content directory specified in the config file through LightWave. Therefore, if you set your content directory properly in LightWave itself, and you have LWSN on the same machine using the same config file as LightWave (which it does by default), you do not need to specify this option in the LWSN command line file, unless you wish to use a content directory that is different than the one currently set in LightWave. You would also need to specify this path if you were running LWSN on a different machine than you are running LightWave.

Please review Config Files: Content Directory, for important information about using LightWave's content directory properly.

## Quiet Mode

-q : Optional: Suppresses terminal output during frame rendering, still reports as each frame finishes.

Normally while LWSN is busy rendering it outputs quite a bit of text information to the terminal interface. This can be useful for monitoring progress and problems such as missing plugins, etc. Once everything is up and running however, all this text writing to the terminal may be unnecessary and may slightly slow down the rendering. The -q parameter turns off all the output during the rendering of an individual frame. You'll still get text output between frames, but not during frame rendering. This may slightly speed up rendering but will also make LWSN less responsive to user input, such as if you wish to save the log or quit LWSN.

Here's what the output for one frame looks like normally, without the -q parameter:

```
LightWave PowerMac ScreamerNet Module (Build 690)
CPU number: 774
Current directory is now "HD:Samples:Content:".
Loading ":Scenes:Toys:ToysTest.lws".
Clearing scene
Loading settings
Loading "Objects/Toys/Ball.lwo"
Loading "Objects/Toys/Blocks.lwo"
Loading "Objects/Toys/Floor.lwo"
Validating scene
Scene loaded.
Allocating frame buffers.
Allocating segment buffers.
Updating geometry.
Moving Ball.
Moving Blocks.
Moving Floor.
Optimizing Ball.
Optimizing Blocks.
Optimizing Floor.
Computing shadow map for Light.
Transforming coordinates.
Removing hidden polygons.
Computing polygon distances.
Sorting polygons.
Rendering frame 1, segment 1/1, pass 1/5.
Rendering transparent polygons.
Integrating pixels.
Rendering frame 1, segment 1/1, pass 2/5.
Rendering transparent polygons.
Integrating pixels.
Rendering frame 1, segment 1/1, pass 3/5.
Rendering transparent polygons.
Integrating pixels.
Rendering frame 1, segment 1/1, pass 4/5.
Rendering transparent polygons.
Integrating pixels.
Rendering frame 1, segment 1/1, pass 5/5.
Rendering transparent polygons.
Integrating pixels.
Freeing segment buffers.
Allocating filter buffer.
Applying soft filter.
Freeing filter buffer.
Allocating segment buffers.
Writing RGB image to HD:Content:Images:TESTS:Toys0001.tga.
Frame completed.
Last Frame Rendered: 1.
Rendering Time: 0.0 seconds.
Freeing segment buffers.
Freeing frame buffers.
Here's what the output looks like with the -q parameter included:
LightWave PowerMac ScreamerNet Module (Build 690)
CPU number: 774
Current directory is now "HD:Samples:Content:".
Loading ":Scenes:Toys:ToysTest.lws".
Scene loaded.
Last Frame Rendered: 1.
```



## LWSN Mac Standalone Mode (-3) cmdLine Syntax

The LWSN standalone mode (-3) specific cmdLine syntax is as follows, all on one line.

```
-3 [-c<config directory>] [-d<content directory>] [-q] <scene file>
<first frame> <last frame> [<frame step>]
```

The first four cmdLine parameters are fully explained above under General LWSN cmdLine Syntax.

### Scene File Path

<scene file>: Path to the scene file. May be a full or a relative path, enclosed in quotes.

Full paths specify the entire path from the hard drive down to the scene file, such as:

```
"YourHD:Users:username:Documents:LWContent:Scenes:MyScene.lws"
```

Relative paths are relative to the current Content directory specified in the command line itself, or from within LightWave's config file. Relative paths must start with a leading colon such as:

```
":Scenes:MyScene.lws"
```

### Frames to Render

<first frame>: First frame to render, may be higher than last frame if step is negative.

<last frame>: Last frame to render, may be lower than first frame if step is negative.

[<frame step>]: Optional: defaults to 1, may be positive or negative.

This specifies which frames to render between the first and last frame. When set to 1, LWSN would render every frame from the first frame, up to the last frame. When set to 2, LWSN would render every other frame from the first frame up to the last frame. If set to -1, LWSN would render every frame from the first frame, down to the last frame and in this case the first frame should be higher than the last frame to render in reverse order.

## LWSN Mac Batch Mode (-2) cmdLine Syntax

The LWSN batch & network mode (-2) specific cmdLine syntax is as follows, all on one line.

```
-2 [-c<config directory>] [-d<content directory>] [-q] [-t<check interval>] <job command file> <acknowledgment file>
```

The first four cmdLine parameters are fully explained above under General LWSN cmdLine Syntax.

## Time Check Interval

-t : Optional: Time check interval in seconds. For instance as follows:

```
-t60
```

This would have LWSN attempt to check the job file every 60 seconds during rendering for status or abort commands. Other commands are ignored.

Normally, once LWSN begins rendering a frame, it does not check the job file for any further instructions until the frame finishes rendering. During the initial setup and testing of your render farm or when testing a new scene, it may be useful to allow LWSN to check the job file for abort commands. Particularly if individual frames take a long time to render. This way, if you notice a problem right off the bat (a missing plugin for instance), you can abort the rendering without having to wait for all your LWSN nodes to finish rendering an entire frame each or force quitting each node.

For instance, if your frames use ray tracing, area lights, radiosity, caustics, high antialiasing levels, motion blur and/or depth of field, they may take a long time to render. In this case you could set -t to 60 seconds, then LWSN would try to check the job file every 60 seconds to see if the job should be aborted or not. During certain processing, LWSN may take longer than 60 seconds before it can check, but it doesn't have to wait for an entire frame to render before being able to stop. Once you are sure everything is running smoothly, it is best to remove this option, so that no extra time is taken checking the job file unnecessarily, especially if you are network rendering many frames.

## Job Command File

<job command file> : job# File & node number for LWSN to read commands, such as:

"job1" or ".job1" Reads the job1 file in the top level of the Content directory.

":Commands:job1" Reads the job1 file in a Commands directory in the Content directory.

```
"YourHD:Applications:LightWave10:Programs:job1"
```

Reads the job1 file in the Programs directory in the LightWave 10 directory, where YourHD should be replaced with the name of your hard drive where your applications are stored.

Make sure that both your job# and ack# both use the same number. This number is what actually determines the CPU number of the LWSN node, not any number that you may also be using in the LWSN and cmdLine file names. Though to keep things from getting confusing I suggest using similar numbers for the LWSN and cmdLine names as well as for the job# and ack# in the cmdLine file itself.

For example:

Application name: LWSN-1

cmdLine Name: LWSN-1 cmdLine

cmdLine Contents: -2 "job1" "ack1"

For additional information please review Config Files: Command Directory.



## Acknowledgement File

<acknowledgement file> : ack# File & node number for LWSN to write replies, such as:

“ack1” or “:ack1” Writes the ack1 file in the top level of the Content directory.

“:Commands:ack1” Writes the ack1 file in a Commands directory in the Content directory.

“YourHD:Applications:LightWave10:Programs:ack1”

Writes the ack1 file in the Programs directory in the LightWave 10 directory, where YourHD should be replaced with the name of your hard drive where your applications are stored.

For additional information please review Config Files: Command Directory above.

## LWSN Mac Command Line Dialog Interface

In addition to running LWSN with a cmdLine file, you can also run it using the default command line dialog interface. If you launch LWSN without a cmdLine file, it will open an input dialog which allows you to type in the command line arguments.

Unfortunately, in LightWave 10 on Mac OS X you can't paste any text into the dialog the way you used to be able to on Mac OS 9. The only real benefit of using this cmdLine dialog, rather than a cmdLine file, is that you can direct LWSN to write its output log to a file, rather than a console window. This can come in handy when trying to troubleshoot your render farm.

To use the command line dialog simply follow these steps.

1. Move or rename the LWSN cmdLine file, so that LWSN won't find it.
2. Launch LWSN which will open the command line dialog.
3. Type in your command line parameters, just like in a cmdLine file.
4. Click on LWSN.out and select a save location when prompted.

To successfully render a scene using LWSN you must first prepare the scene correctly, before submitting it to be rendered. The first thing to verify is that your scene files are properly structured in a self contained Content Directory as explained previously.

### •Setting a Scene's Render Options

oFrame Range to Render

oRender Output Destination

### •LWSN Spreadsheet BUG

## Frame Range to Render

When rendering scenes with LightWave or with LWSN in batch or network mode (-2), you must set the range of frames to render directly within the scene itself. When rendering scenes with LWSN in standalone mode (-3) or when using a third party network controller you may override the scene's render range setting at the time of rendering.

Follow these steps to set the frame range to render for a scene:

1. Launch LightWave.
2. Load the desired scene with: Load->Scene
3. Open the Render Options panel with: Render->Options->Render Options
4. Set the following fields:

o Render First Frame: The first frame to render.

o Render Last Frame: The last frame to render.

o Render Frame Step: Frame increment to render.

For example:

1 renders every frame starting with the first frame, up to the last frame.

2 renders every other frame starting with the first frame, up to the last frame.

5 renders every fifth frame from the first frame to the last frame.

-1 renders every frame in reverse order, from the first frame down to the last frame, in this case the first frame would be set higher than the last frame.

5. Save the scene file.

## Setting a Scene's Render Options

Once your scene is properly structured in a self contained Content Directory, you need to open it in LightWave to set a number of render settings.



## Render Output Destination

You must set the scene's render output destination within the scene file itself, or none of the rendered frames will be saved anywhere. You must set the render output prefix file name, file format and destination path in the scene file itself within LightWave, before rendering the scene with LWSN.

LWSN only renders individual frames, it does not render animations in formats such as QuickTime. Instead, you render your animations to still frames and then use QuickTime Player, After Effects, Final Cut Pro or another program to load the image sequence, and save it as a QuickTime movie.

Follow these steps to set the render output settings for a scene:

1. Launch LightWave.
2. Load the desired scene with: Load->Scene
3. Open the Render Options panel with: Render->Options->Render Options
4. Click the Output Files tab.
5. Locate the Save RGB section of the Output Files area.
6. Click the RGB Files button.
7. Type a file name prefix in the Save As field of the RGB Prefix file dialog. This can be any name you wish. LightWave will add a frame number and file extension to each rendered image file. In this example we'll use the word RenderFileName as the RGB prefix.
8. Navigate to the directory you wish to use to save the rendered frames.

LightWave defaults to use the Images directory inside the Content directory.

It's usually a good idea to make a new folder inside the images folder to keep all the separate rendered frames for a single scene together. Then use this new folder as the output destination directory.



NOTE: All users running LWSN must have read/right access to this output directory.

9. Click the Save button to dismiss the RGB Prefix dialog and accept the changes.
10. Set the RGB Type pop-up field to any desired file type, such as LW\_TGA32(.tga)
11. Set the Output Filename Format to any naming format you wish. This tells LWSN how to name each frame image. The names are built from the RGB Prefix you typed followed by a frame number and optional filename extension. In this example we'll use Name\_001.xxx.
12. You should now see RenderFileName\_001.tga, ... in the readout field next to the RGB Files button. This is how the rendered image files will be named inside the output directory.
13. Save the scene file.

## Standalone Mode

LWSN may be used in standalone mode (-3) to render scenes independent of LightWave itself. This method may be used to render a scene in the background while you work on another scene in LightWave, which is especially useful on dual processor machines. It may also be used to render a scene on another machine that doesn't have LightWave installed, as long as you can access the LightWave machine with your configs, programs and content over your network via file sharing, or copy the files all to the other machine before rendering.

### LWSN Mac Standalone Mode (-3) cmdLine Syntax

You tell LWSN which mode to use, as well as what to render through the use of the LWSN cmdLine file. The LWSN standalone mode (-3) cmdLine syntax is as follows, all on one line.

```
-3 [-c<config directory>] [-d<content directory>] [-q] <scene file>
<first frame> <last frame> [<frame step>]
```

For an in-depth discussion of LightWave's cmdLine files, complete with an explanation of each command line parameter, please review Mastering LightWave's Mac OS X cmdLine Files.



## LWSN Mac Standalone Command Line Examples

The following command line examples must be typed all on one line.

```
-3 ":Scenes:MyScene.lws" 1 1
```

Uses the default config file in "YourHD:Users:username:Library:Preferences"

Uses the content directory specified in the config file, set in LightWave before launching LWSN.

Renders the file MyScene.lws in the Scenes directory inside the content directory.

Renders from frame 1 to frame 1. Renders only 1 frame.

```
-3 -c "YourHD:Users:username:Library:Preferences"
":Scenes:MyScene.lws" 1 10
```

Uses the config file in "YourHD:Users:username:Library:Preferences"

Uses the content directory specified in the config file, set in LightWave before launching LWSN.

Renders the file MyScene.lws in the Scenes directory inside the content directory.

Renders every frame (since no step was specified), from frame 1 to 10. Renders 10 frames.

```
-3 -c "YourHD:Users:username:Library:Preferences" -d "YourHD:Users:username:Documents:LWContent" ":MyScene.lws" 1 10 2
```

Uses the config file in "YourHD:Users:username:Library:Preferences"

Uses the content directory "YourHD:Users:username:Documents:LWContent"

Renders the file MyScene.lws inside the content directory.

Renders every other frame (step 2), from frame 1 to 10. Renders 5 frames, 1, 3, 5, 7 & 9.

```
-3 -c "YourHD:Users:username:Library:Preferences" -d "YourHD:Users:username:Documents:LWContent" "YourHD:Users:username:Documents:LWContent:MyScene.lws" 10 1 -1
```

Uses the config file in "YourHD:Users:username:Library:Preferences"

Uses the content directory "YourHD:Users:username:Documents:LWContent"

Renders the file "YourHD:Users:username:Documents:LWContent:MyScene.lws"

Renders every frame in reverse order (step -1), from frame 10 down to 1. Renders 10 frames.

```
10, 9, 8, 7, 6, 5, 4, 3, 2, 1
```

## Step-by-step LWSN Standalone Rendering Example

Here's a step-by-step example of how to render one of the included LightWave sample scenes using LWSN in standalone mode (-3) on the Macintosh.

### Setting up the Config File

In this example we are using the default Mac OS X config location of: "YourHD:Users:username:Library:Preferences"

1. Launch LightWave.

2. Set the Content Directory with Edit->Set Content Directory...

"YourHD:Applications:LightWave10:Content:Classic Content"

This assumes you are running Mac OS X as an administrator with write access to the content directory inside the Applications directory. Otherwise copy the Classic Content directory to your home Documents directory and set the content directory in LightWave to: "YourHD:Users:username:Documents:Classic Content"

3. Open Render->Options->Render Options and set Multithreading to 1 Thread.

4. Open the Camera Properties panel and set the Segment Memory Limit to 32.

5. Click the yes button when asked if this should be the new default setting.

6. Quit LightWave to save the config file.

### Setting up the LWSN cmdLine File.

1. Open the LWSN cmdLine file located as follows:

"YourHD:Applications:LightWave10:Programs:LWSN cmdLine"

2. Edit its contents to read:

```
-3 ":Scenes:Characters:DancingTeapot.lws" 1 5
```

3. Save the cmdLine file, making sure no file extension is added to the filename.

Setting up the Scene for Rendering

1. Launch LightWave.

2. Use Items->Load->Scene to load: ":Scenes:Characters:DancingTeapot.lws"

3. Open Render->Options->Render Options

4. Click the Output Files tab.

5. Click the RGB Files button.

6. Type DancingTeapot in the Save As: field.

7. Navigate to the Images directory inside the Content Directory.

8. Create a New Folder named DancingTeapotRenders.

9. Click the Save button to dismiss the dialog and accept the changes.

10. Set the RGB Type pop-up menu to LW\_TGA24(.tga)

11. The RGB readout should now read DancingTeapot0001.tga, ...

12. Save the scene.

13. Quit LightWave.





## Rendering the Scene with LWSN

1. Launch LWSN
2. LWSN should now render frames 1 through 5 of DancingTeapot.lws and place them in the Classic Content:Images:DancingTeapotRenders folder.

## Batch Rendering

Batch rendering enables you to queue up and render multiple scenes all at once, unattended. Batch rendering is similar to network rendering except the same machine is used as both the host and the rendering node, without a network. In the case of multiprocessor Macs, you can run multiple copies of LWSN on the same machine, one for each available processor.

## LWSN Mac Batch Mode (-2) mdLine Syntax

You tell LWSN which mode and settings to use, through the use of the LWSN cmdLine file. The LWSN batch & network mode (-2) cmdLine syntax is as follows, all on one line.

```
-2 [-c<config directory>] [-d<content directory>] [-q] [-t<check interval>] <job command file> <acknowledgment file>
```

For an in-depth discussion of LightWave's cmdLine files, complete with an explanation of each command line parameter, please review Mastering LightWave's Mac OS X cmdLine Files.

## LWSN Mac Batch Command Line Examples

On dual processor Macs you may run multiple instances of LWSN, one for each CPU. To run more than one copy of LWSN you need to duplicate the LWSN application and give each a unique name such as: LWSN-1 and LWSN-2. You would then create matching command line files named: LWSN-1 cmdLine and LWSN-2 cmdLine. Both of these command line files would contain an identical command line except the LWSN-1 version would reference job1 & ack1 and the LWSN-2 version would reference job2 & ack2. It's the job & ack numbers inside the command line file that actually tell LWSN and the Network Rendering panel which CPU node is which. It's useful to use the same numbers in the file names as well, to help you identify which running LWSN node is which.

The following command line examples must be typed all on one line.

```
-2 ".job1" ".ack1"
```

Uses the default config location of: "YourHD:Users:username:Library:Preferences"

Uses the Content directory specified in the config file, set in LightWave before launching LWSN.

Uses the job1 and ack1 files inside the top level of the Content directory.

This instance of LWSN would be designated CPU node 1.

```
-2 -c"YourHD:Applications:LightWave10:Configs" ".job1" ".ack1"
```

Uses a config file in "YourHD:Applications:LightWave10:Configs"

Uses the content directory specified in the config file, set in LightWave before launching LWSN.

Uses the job1 and ack1 files inside the top level of the Content directory.

This instance of LWSN would be designated CPU node 1.

```
-2 -c"YourHD:Applications:LightWave10:Configs" -d"YourHD:Users:username:Documents:LWContent" ".job1" ".ack1"
```

Uses a config file in "YourHD:Applications:LightWave10:Configs"

Uses the content directory "YourHD:Users:username:Documents:LWContent"

Uses the job1 and ack1 files inside the top level of the Content directory.

This instance of LWSN would be designated CPU node 1.

```
-2 -c"YourHD:Applications:LightWave10:Configs" -d"YourHD:Users:username:Documents:LWContent" ".Commands:job2" ".Commands:ack2"
```

Uses a config file in "YourHD:Applications:LightWave10:Configs"

Uses the content directory "YourHD:Users:username:Documents:LWContent"

Uses the job2 and ack2 files inside a Commands directory inside the Content directory.

This instance of LWSN would be designated CPU node 2.





## LightWave's Built-in Network Render Controller

LightWave comes with a very basic Network Rendering control panel that enables you to perform basic batch and network rendering. It is sufficient for occasional use, but if you intend to do heavy duty network rendering in a production environment, you'll find the investment in a third party network controller well worthwhile. We use RenderFarm Commander and ScreamerNet Controller for Mac OS X to run our in-house DreamLight render farm and will cover their use later when discussing advanced network rendering.

The built-in controller is sufficient for simple batch rendering. It's also useful to use when troubleshooting a third party controller. If you're having trouble using a third party controller, you can try a render test with the built-in controller to make sure your LWSN render farm is set up properly. I'll cover the built-in network controller's use here and use it for the batch rendering example to follow.

## Running the Network Rendering Control Panel

The Network Rendering control panel is opened in LightWave 10 by selecting:

Render->Utilities->Network Render.

When the Network Rendering panel is open and rendering, LightWave can't be used to do anything else, which tends to defeat the purpose of performing a batch or network render in the first place... A trick you can use to get around this limitation is to make a duplicate copy of LightWave and rename it LightWave-SN. Then you can use the copy called LightWave-SN to control your batch rendering and still launch the original LightWave to work on something else while the batch render continues. When doing this you should also make a LightWave-SN cmdLine file which contains at least the -0 (dash zero) switch to disable the Hub. That way you won't have two copies of LightWave both trying to synchronize things through the Hub, which could cause problems.

## Render Node Controls

**Command Directory:** The directory where the LWSN job# & ack# communication files will be stored. The Network Rendering controller doesn't actually contact the LWSN nodes themselves, it simply creates job# files with simple commands in them. LWSN nodes constantly poll for their job files and execute the commands they find. They then write a reply into an associated ack# file which the Network Rendering controller continuously polls to monitor the nodes.

**Maximum CPU Number:** The number of LWSN nodes to attempt to control.

**Screamer Init:** Initializes all the currently running LWSN nodes.

**ScreamerNet Status:** The text field to the right of the Screamer Init button shows the current ScreamerNet status.

**CPU list:** Lists all the currently running LWSN render nodes.

**Screamer Render:** Starts the LWSN nodes rendering the first scene.

**Cancel Render (Escape Key):** Stops the Network Rendering controller from issuing new render commands. With LightWave 10, it will attempt to interrupt renderings in progress so the user does not have to wait till each node finishes rendering a complete frame first. You may only cancel a render when the ScreamerNet status reads "Waiting for CPUs to finish rendering." If you need to cancel a render during any other time, for instance during a crashed LWSN scene load, you'll unfortunately have to force quit LightWave with Command-Option-Esc. Then quit all remaining LWSN nodes. You may then start everything back up again and try again... Third party render controllers are not as limited.

**Screamer Shutdown:** Quits all LWSN nodes, when they finish rendering their current frames. To quit a LWSN node before it finishes rendering a frame, bring it to the foreground and press command-period. If you try to quit LWSN 7.5 using the menu LWSN->Quit or by typing command-q, it will crash. This bug has been fixed in LWSN 10.0.

## Scene List Controls

The scene list in the Network Rendering panel can not be modified while anything is currently rendering. This is the single greatest limitation of the built-in controller. You must stop any renders in progress before you can add or remove scenes from the list, which seriously cripples the built-in controller's usefulness. To stop renders in progress you press Cancel Render or the Escape key.

**Clear List:** Clears all scenes from the scene list.

**Add Scene to List:** Adds a scene to the end of the scene list.

**Remove Scene:** Removes the scene located at the top of the list display. To delete a specific scene, drag the scroll bar until the scene you wish to delete is located at the top of the list display. Then press the Remove Scene button.

Before adding a scene to the scene list you must make sure it is properly set up as explained previously. You cannot modify the frames to render, the output destination or even the order of the list. Now you can probably see why I say the built-in controller is rather limited for a production environment...



## Step-by-step LWSN Batch Rendering Example

Here's a step-by-step example of how to render a batch of included LightWave sample scenes using LWSN in batch / network mode (-2) on the Macintosh. This example uses two LWSN instances running on a dual processor Power Macintosh G5. On a single processor machine, or if you wish to batch render on one processor while continuing to work on the other, simply run one instance of LWSN.

### Setting up the Config File

In this example we are using the default Mac OS X config location of: "YourHD:Users:username:Library:Preferences"

For more information see Mastering LightWave's All Important Config Files.

1. Launch LightWave.
2. Set the Content Directory with Edit->Set Content Directory...
- "YourHD:Applications:LightWave10:Content:Classic Content"

This assumes you are running Mac OS X as an administrator with write access to the content directory inside the Applications directory. Otherwise copy the Classic Content directory to your home Documents directory and set the content directory in LightWave to: "YourHD:Users:username:Documents:Classic Content"

See Content Directory for more information.

3. Open the Render->Utilities->Network Render panel.
4. Click the Command Directory button.
5. Navigate to the same Classic Content directory you set in step #2.
6. Click New Folder to create a new folder in the Content directory and name it Commands.
7. While inside the Commands folder, click the Choose button to close the dialog and accept the new command directory.
8. When asked to initialize ScreamerNet, click No.
9. Set the Maximum CPU Number to 2.
10. Open Render->Options->Render Options and set Multithreading to 1 Thread.

See Multithreading for more information.

11. Open the Camera Properties panel and set the Segment Memory Limit to 32.

See Default Segment Memory for more information.

12. Click the yes button when asked if this should be the new default setting.
13. Quit LightWave to save the config file.

### Setting up the LWSN Nodes and cmdLine Files

For more information see Mac OS cmdLine Files.

1. Open the directory "YourHD:Applications:LightWave10:Programs" and locate the LWSN application.
2. Make two copies of LWSN and name them LWSN-01 and LWSN-02, with no file extension.
3. Make one copy of the file LWSN cmdLine and name it LWSN-01 cmdLine, with no file extension.
4. Edit the LWSN-01 cmdLine file's contents to read:  
-2 ":Commands:job1" ":Commands:ack1"
5. Save the cmdLine file, making sure no file extension is added to the filename.
6. Duplicate the LWSN-01 cmdLine file and name it LWSN-02 cmdLine.
7. Edit the LWSN-02 cmdLine file's contents to read:  
-2 ":Commands:job2" ":Commands:ack2"
8. Save the cmdLine file, making sure no file extension is added to the filename.

### Launching the LWSN Nodes

1. Launch LWSN-01.
2. It should identify itself as CPU number: 1 in the console.
3. Launch LWSN-02, launch them one at a time or you may get an error.
4. It should identify itself as CPU number: 2 in the console.
5. They may say Can't open job file ":Commands:job#", this is normal until LightWave creates the job files.



## Setting up the Scenes for Batch Rendering

For more information see Successful Scene File Preparation for LWSN.

1. Launch LightWave.
2. Use Items->Load->Scene to load: `“:Scenes:Aviation:MustangLowRes.lws”`
  1. Open Render->Options->Render Options
  2. Verify the following render range fields:
    - ☐ Render First Frame: 1
    - ☐ Render Last Frame: 60
    - ☐ Render Frame Step: 1
  3. Click the Output Files tab.
  4. Click the RGB Files button.
  5. Type MustangLowRes in the Save As: field.
  6. Navigate up one level out of the Images directory to the Content Directory.
  7. Create a New Folder named Renders, or navigate into it if you already have one.
  8. Create a New Folder inside Renders, named MustangLowRes.
  9. Click the Save button to dismiss the dialog and accept the changes.
  10. Set the RGB Type pop-up menu to LW\_TGA24(.tga)
  11. The RGB readout should now read MustangLowRes0001.tga, ...
  12. Save the scene.
3. Use Items->Load->Scene to load: `“:Scenes:Aviation:RC_Helicopter.lws”`
  1. Open Render->Options->Render Options
  2. Verify the following render range fields:
    - ☐ Render First Frame: 1
    - ☐ Render Last Frame: 20
    - ☐ Render Frame Step: 1
  3. Click the Output Files tab.
  4. Click the RGB Files button.
  5. Type RC\_Helicopter in the Save As: field.
  6. Navigate to the Render folder you already created inside the Content Directory.
  7. Create a New Folder inside Renders, named RC\_Helicopter.
  8. Click the Save button to dismiss the dialog and accept the changes.
  9. Set the RGB Type pop-up menu to LW\_TGA24(.tga)
  10. The RGB readout should now read RC\_Helicopter0001.tga, ...
  11. Save the scene.
4. Quit LightWave.

## Rendering the Batch with LightWave's

## Built-in Controller

For more information see Using LightWave's Built-in Network Rendering Controller.

1. Launch LightWave.
2. Open the Render->Utilities->Network Render panel.
3. Set the Maximum CPU Number to 2.
4. Click the Screamer Init button.
5. Click OK when the dialog appears stating: 2 available ScreamerNet CPUs were detected. You should now see both LWSN nodes repeatedly writing LightWave command: wait. and sendack: Ready to the console. You'll also see two CPU's listed in the Network Rendering panel with a status of Ready.
6. Click the Add Scene to List button and add the scene: `“:Scenes:Aviation:MustangLowRes.lws”`
7. Click the Add Scene to List button again and add the scene: `“:Scenes:Aviation:RC_Helicopter.lws”`
8. Hit the Screamer Render button and the LWSN nodes will begin rendering frames.
9. As each frame is finished, it will be written to the output directory that was set in each scene. `“:Renders:MustangLowRes”` & `“:Renders:RC_Helicopter”`
10. If you wish to stop the rendering before it is finished, press the Escape key while the Network Rendering panel is in the foreground, then hit the Screamer Shutdown button. This will stop rendering new frames and will quit the LWSN nodes once they finish rendering their current frame. If a LWSN node crashes during loading a scene, the Network Rendering panel will remain waiting for the scene to load and will never recover. In such an instance you must force Quit LightWave with Command-Option-Esc. To quit a LWSN node before it finishes rendering a frame, bring it to the foreground and press command-period.

Basic network rendering with LWSN enables you to harness your entire network of powerful Macintosh computers to render your LightWave scenes much faster than would be possible on a single machine. Network rendering with LWSN is almost exactly the same as batch rendering, described previously, which you should review before proceeding with the following network tutorial. The only difference between network and batch rendering is that some of the LWSN nodes run on additional Macs on your network, rather than on a single machine.

- Setting up the Mac OS X Network for Rendering
  - oPreparing the Hard Drives for Basic Network Rendering
  - oSharing the Host Machine on the Network
  - oMounting the Host Machine from the Render Machines
- Step-by-step LWSN Mac Basic Network Rendering Example
  - oSetting up the Host Config File
  - oSetting up the LWSN Nodes and cmdLine Files
  - oLaunching the LWSN Nodes on the Remote Rendering Machines
  - oSetting up the Scenes for Batch/Network Rendering
  - oRendering the Network Batch with LightWave's Built-in Controller



## Setting up the Mac OS X Network for Rendering

In order to perform network rendering, you'll obviously need a functioning network. This tutorial assumes that you have at least a basic peer-to-peer Mac OS X network already up and running which allows you to connect from one Mac to another to share files. You don't need Mac OS X Server, any version of Mac OS X is capable of peer-to-peer networking. If you don't already have a physical network, you'll first need to set one up. You can find information about setting up a Mac OS X network with this Google search.

This tutorial was written using Mac OS X.3.3 running on a 100 bps Ethernet network and LightWave 8. Other versions of Mac OS X and/or LightWave may be slightly different, but the basic concepts will remain the same...

There was a bug in Mac OS X.2.6 that sometimes prevented more than four LWSN instances from sharing the same dynamic loading libraries and/or plug-ins successfully over the network. The symptoms were that the fifth and higher LWSN nodes sometimes crashed on launch (warning about missing various libraries that were really there), or crashed when loading a scene. This bug appears to be partially fixed in Mac OS X.2.8, though I have still seen nodes higher than four sometimes crash on loading scenes, fail to save frames and fail to load textures. This bug appears to have been fixed in Mac OS X.3.3. However, for best results with more than four LWSN nodes, I still prefer to install separate copies of LWSN, shared libraries and all plug-ins on each remote rendering machine as outlined later in Advanced Network Rendering with LWSN Mac.

## Preparing the Hard Drives for Basic Network Rendering

The simplest way to set up network rendering on Mac OS X entails logging into the host machine (the Mac with the LightWave applications and possibly the Content Directory) as the same administrative user from each remote network machine. Then running the LWSN nodes on the remote rendering machines, from the host machine across the network. This way there is only one set of configs, programs, shared libraries and plug-ins. As the host machine's administrator you then have full read/write access to everything you need on the host machine. You'll be logging into the host machine from each remote rendering machine and mounting the host machine's main hard drive on the desktop of each render machine. This means that all your hard drives must have unique names, or LightWave won't know which hard drive to access, the local drive or the remote drive.

Therefore, the first step to set up your network for rendering is to simply give a unique name to each hard drive on each machine you wish to use as either a host or render machine. If they don't already have unique names that is. For example you could name the main hard drives for each machine as follows: HostHD, Render01HD, Render02HD, Render03HD or anything else that keeps each hard drive name unique.

## Sharing the Host Machine on the Network

The next step in setting up your network rendering is to make sure the host machine, which contains your LightWave applications and Content Directory, is available on the network for the other remote rendering machines to log into.

Simply follow these steps to verify that your host machine is available:

1. Launch System Preferences from the Apple menu or the Dock.
2. Click the Sharing icon under Internet & Networking.
3. Click the Services tab.
4. Select Personal File Sharing and click the Start button, if it's not already on.
5. Click the Firewall tab and click Start if the firewall was not already on. This helps secure your Mac.
6. In some cases, such as if you're running any older Mac OS 9 systems as render nodes (with LWSN 7.5), it may also be necessary to activate AppleTalk.

1. Click the Network icon at the top or under Internet & Networking.
2. Select Built-in Ethernet on the Show: pop-up menu.
3. Click the AppleTalk tab.
4. Turn on the Make AppleTalk Active checkmark.

## Mounting the Host Machine from the Render Machines

Now you need to mount the host machine's main hard drive onto the desktop of each of the remote render machines. Simply follow these steps on each remote rendering machine:

1. In any Finder window, click the Network icon.
2. Locate and select the host machine on the network, it may show up directly or you may need to open a subgroup depending upon how your network is configured.
3. Once the host machine is selected, click the Connect button.
4. When the log-in screen appears click the Registered User button. Guest access won't work in this instance since you wouldn't have sufficient read/write privileges.
5. Enter the username and password of the main administrative user of the host machine, not necessarily the remote rendering machine, unless they are the same. You must log into the host as the host administrator to mount the host machine's main hard drive. Click the Connect button when ready.
6. You'll now get a dialog box showing all available home directories as well as all hard drives connected to the host computer. Select the main hard drive of the host computer where your LightWave applications and Content Directory are located. Click OK and the hard drive will be mounted on the desktop of the remote rendering node.



## Step-by-step LWSN Mac Basic Network

### Rendering Example

Here's a step-by-step example of how to perform basic network rendering of included LightWave sample scenes using LWSN in network render mode (-2) on the Macintosh with the built-in Network Rendering controller. This example uses four LWSN instances running on two Dual Processor Power Macintosh G4's. One instance of LWSN is run for each available CPU. A third Power Macintosh host machine is left free to handle the network controller. For best results with more than four LWSN nodes, I prefer to use the Advanced Network Rendering with LWSN Mac setup.

### Setting up the Host Config File

In this example we are using the default Mac OS X config location of: "HostHD:Users:adminUserName:Library:Preferences"

For more information see Mastering LightWave's All Important Config Files.

1. Launch LightWave.
2. Set the Content Directory with Edit->Set Content Directory...  
"HostHD:Applications:LightWave10:Content:Classic Content"  
This assumes you are running Mac OS X as an administrator with write access to the content directory inside the Applications directory. Otherwise copy the Classic Content directory to your home Documents directory and set the content directory in LightWave to: "HostHD:Users:adminUserName:Documents:Classic Content"  
See Content Directory for more information.
3. Open the Render->Utilities->Network Render panel.
4. Click the Command Directory button.  
See Command Directory for more information.
5. Navigate to the same Classic Content directory you set in step #2.
6. Click New Folder to create a new folder in the Content directory and name it Commands.
7. While inside the Commands folder, click the Choose button to close the dialog and accept the new command directory.
8. When asked to initialize ScreamerNet, click No.
9. Set the Maximum CPU Number to 4.
10. Open Render->Options->Render Options and set Multithreading to 1 Thread.  
See Multithreading for more information.
11. Open the Camera Properties panel and set the Segment Memory Limit to 32.  
See Default Segment Memory for more information.
12. Click the yes button when asked if this should be the new default setting.
13. Quit LightWave to save the config file.

## Setting up the LWSN Nodes and cmdLine Files

For more information see Mastering LightWave's Mac OS cmdLine Files.

1. Open the directory "HostHD:Applications:LightWave10:Programs" and locate the LWSN application.
2. Make four copies of LWSN and name them LWSN-01 through LWSN-04, with no file extension.
3. Make one copy of the file LWSN cmdLine and name it LWSN-01 cmdLine, with no file extension.
4. Edit the LWSN-01 cmdLine file's contents to read, all on one line:  
-2 -c"HostHD:Users:adminUserName:Library:Preferences"  
":Commands:job1" ":Commands:ack1"



Note: you need to specify the path to the config directory on the Host machine, since each remote LWSN node will be using the same config file over the network.

5. Save the cmdLine file, making sure no file extension is added to the filename.
6. Duplicate the LWSN-01 cmdLine file three times and name them LWSN-02 cmdLine, LWSN-03 cmdLine, and LWSN-04 cmdLine.
7. Edit each of the the LWSN cmdLine file's contents by changing the job and ack number to match the file name's number, as follows:  
LWSN-02 cmdLine contents:  
-2 -c"HostHD:Users:adminUserName:Library:Preferences"  
":Commands:job2" ":Commands:ack2"  
LWSN-03 cmdLine contents:  
-2 -c"HostHD:Users:adminUserName:Library:Preferences"  
":Commands:job3" ":Commands:ack3"  
LWSN-04 cmdLine contents:  
-2 -c"HostHD:Users:adminUserName:Library:Preferences"  
":Commands:job4" ":Commands:ack4"
8. Save each cmdLine file, making sure no file extension is added to the filename.





## Launching the LWSN Nodes on the Remote Rendering Machines

For additional information review Sharing the Host Machine & Mounting the Host Machine from the Render Machines.

1. On each of the remote Macs, in any Finder window, click the Network icon.
2. Select and Connect to the host machine on the network.
3. Log into the host machine as the main administrative user of the host machine.
4. Select the host machine's main hard drive to mount it on the remote desktop.
5. Open the "HostHD:Applications:LightWave10:Programs" folder.
6. Launch the next two available LWSN-0# nodes, one at a time, on each dual processor remote machine as follows.
  - o Remote Render Machine 01: LWSN-01 & LWSN-02
  - o Remote Render Machine 02: LWSN-03 & LWSN-04

## Setting up the Scenes for Batch/Network Rendering

For more information see Successful Scene File Preparation for LWSN.

1. Launch LightWave.
2. Use Items->Load->Scene to load:
  - “.Scenes:Aviation:MustangLowRes.lws”
  1. Open Render->Options->Render Options
  2. Verify the following render range fields:
    - ☐Render First Frame: 1
    - ☐Render Last Frame: 60
    - ☐Render Frame Step: 1
  3. Click the Output Files tab.
  4. Click the RGB Files button.
  5. Type MustangLowRes in the Save As: field.
  6. Navigate up one level out of the Images directory to the Content Directory.
  7. Create a New Folder named Renders, or navigate into it if you already have one.
  8. Create a New Folder inside Renders, named MustangLowRes.
  9. Click the Save button to dismiss the dialog and accept the changes.
  10. Set the RGB Type pop-up menu to LW\_TGA24(.tga)
  11. The RGB readout should now read MustangLowRes0001.tga, ...
  12. Save the scene.

### 3. Use Items->Load->Scene to load: “.Scenes:Aviation:RC\_Helicopter.lws”

1. Open Render->Options->Render Options
2. Verify the following render range fields:
  - ☐Render First Frame: 1

- ☐Render Last Frame: 60
  - ☐Render Frame Step: 1
3. Click the Output Files tab.
  4. Click the RGB Files button.
  5. Type RC\_Helicopter in the Save As: field.
  6. Navigate to the Render folder you already created inside the Content Directory.
  7. Create a New Folder inside Renders, named RC\_Helicopter.
  8. Click the Save button to dismiss the dialog and accept the changes.
  9. Set the RGB Type pop-up menu to LW\_TGA24(.tga)
  10. The RGB readout should now read RC\_Helicopter0001.tga, ...
  11. Save the scene.
4. Quit LightWave.

## Rendering the Network Batch with LightWave's Built-in Controller

For more information see Using LightWave's Built-in Network Rendering Controller.

1. Launch LightWave.
2. Open the Render->Utilities->Network Render panel.
3. Verify that the Maximum CPU Number is set to 4.
4. Click the Screamer Init button.
5. Click OK when the dialog appears stating: 4 available ScreamerNet CPUs were detected. You should now see both LWSN nodes repeatedly writing LightWave command: wait. and sendack: Ready to the console. You'll also see two CPU's listed in the Network Rendering panel with a status of Ready.
6. Click the Add Scene to List button and add the scene:
  - “.Scenes:Aviation:MustangLowRes.lws”
7. Click the Add Scene to List button again and add the scene:
  - “.Scenes:Aviation:RC\_Helicopter.lws”
8. Hit the Screamer Render button and the LWSN nodes will begin rendering frames.
9. As each frame is finished, it will be written to the output directory that was set in each scene. “.Renders:MustangLowRes” & “.Renders:RC\_Helicopter”
10. If you wish to stop the rendering before it is finished, press the Escape key while the Network Rendering panel is in the foreground, then hit the Screamer Shutdown button. This will stop rendering new frames and will quit the LWSN nodes once they finish rendering their current frame. If a LWSN node crashes during loading a scene, the Network Rendering panel will remain waiting for the scene to load and will never recover. In such an instance you must force Quit LightWave with Command-Option-Esc. To quit a LWSN node before it finishes rendering a frame, bring it to the foreground and press command-period.





There are a few key differences between the basic network setup (which you should review before proceeding) and the advanced network setup that follows. Rather than running all the LWSN instances from the host Mac (which may become unstable for more than four nodes, especially in some versions of Jaguar), we will install separate sets of everything LWSN needs, on each remote rendering Mac, including a separate set of config files. We will then make a content directory available that all LWSN nodes may share, without using administrator access to the host, which would pose a potential security risk. We will finally use a third party network controller to manage all the LWSN nodes, such as Jonathan Baker's robust ScreamerNet Controller for OS X or Bruce Rayne's excellent RenderFarm Commander. Adding either of these network controllers to your LightWave toolbox will give you a very robust Mac OS X network render farm that should be able to handle anything you can throw at it. This advanced setup is not intended to be the quickest or easiest way to network render with LWSN. It's intended to show you how to really master LWSN to set up and manage a robust production level render farm on the Mac.

- Setting up the Mac OS X Network for Rendering
  - oSharing the Host Machine on the Network
  - oSharing the Content Folder with SharePoints
  - oMounting the Content Folder from the Render Machines
- Step-by-step LWSN Mac OS X Advanced Network Rendering Example
  - oCreating a New set of Config Files for ScreamerNet
  - oSetting up the Host Config File
  - oSetting up the Config Files on the First Remote Render Node
  - oSetting up LWSN and the cmdLine Files on the First Remote Node
  - oSetting up LWSN on the Remaining Render Nodes
  - oSetting up the Scenes for Batch/Network Rendering
  - oLaunching the LWSN Nodes on the Rendering Machines
  - oRendering the Network Batch with ScreamerNet Controller.
  - oRendering the Network Batch with RenderFarm Commander.

## Setting up the Mac OS X Network for Rendering

In order to perform network rendering, you'll obviously need a functioning network. This tutorial assumes that you have at least a basic peer-to-peer Mac OS X network already up and running which allows you to connect from one Mac to another to share files. You don't need Mac OS X Server, any version of Mac OS X is capable of peer-to-peer networking. If you don't already have a physical network, you'll first need to set one up. You can find information about setting up a Mac OS X network with this Google search.

This tutorial was written using Mac OS X.3.3 running on a 100 bps Ethernet network and LightWave 8. Other versions of Mac OS X and/or LightWave may be slightly different, but the basic concepts will remain the same...

## Sharing the Host Machine on the Network

The first step in setting up your network rendering is to make sure the host machine, which contains your LightWave applications, third party render controller and Content Directory, is available on the network for the other remote rendering machines to log into.

Simply follow these steps to verify that your host machine is available:

1. Launch System Preferences from the Apple menu or the Dock.
2. Click the Sharing icon under Internet & Networking.
3. Click the Services tab.
4. Select Personal File Sharing and click the Start button, if it's not already on.
5. If you'll be using any PC render nodes in your render farm, you should also activate Windows Sharing.
6. Click the Firewall tab and click Start if the firewall was not already on. This helps secure your Mac.
7. In some cases, such as if you're running any older Mac OS 9 systems as render nodes (with LWSN 7.5), it may also be necessary to activate AppleTalk.

In order to successfully render scenes with LWSN, it's critical that you understand and use the Content Directory properly. For more information see Mastering LightWave's Content Directory.

When rendering scenes over the network, there is an added twist. Not only do all the render nodes need to have access to and be able to locate the shared content folder, they may need to locate it with two different paths to maintain a secure network. This is especially the case when you are using LWSN instances running on remote machines as well as on the host machine at the same time or if you are using a mixture of Macintosh and PC machines.

Typically the path to locate a folder on the host machine locally will be different than the path used to access it remotely, unless the remote user is logging into the host machine using the host machine's administrator user. This however gives the remote machine full, unrestricted access to the host machine, which is not very secure. We'll be using a third party render controller that will seamlessly handle multiple paths to the content directory, based upon where it's being accessed from.

Another potential problem with the shared content directory is that in a production environment a large deal of space is needed to hold all the content and rendered frames. This often means that the content folder needs to be located on a hard drive other than the host machine's internal system drive. On Mac OS X however, the only user that can typically access and mount an external drive, is the host machine's administrator. To solve this problem, we'll be using the free Mac OS X utility SharePoints, by HornWare, to publish our content directory from an external drive. The latest version of SharePoints can always be found on HornWare's site, the version on Apple's site was somewhat outdated the last time I checked. If you do use this utility frequently, a small donation to the author would be appreciated and encouraged.



The following steps will walk you through the process of sharing the content folder with SharePoints.

1. Move your properly structured content folder to wherever you wish it to reside for sharing. In our case it's at the top level of one of our external drives.
2. Rename this folder NetContent. You can call it anything you wish, but I've found that using a generic name like NetContent and leaving it in the same place makes the most sense. This way I can have the entire render farm always accessing this same folder as the content directory. Then when I wish to render a scene on the farm, I simply move all the scene's contents into this NetContent folder (and move older stuff out if I wish). This way, its always accessible to the network, all the LWSN config files will know where to find it and I don't have to keep messing with the config files or SharePoints once everything is set up.
3. Create a new folder named Commands inside the NetContent folder. This will be the command directory for the network controller to communicate with the render nodes.
4. Create another new folder inside the NetContent folder. Name this new folder Renders. This will serve as the render destination for all the remote render nodes.
5. Download the latest version of SharePoints.
6. Install it by copying the SharePoints.prefPane file into your home folder's Library/PreferencePanels folder. Create the PreferencePanels folder in your Library folder if it doesn't already exist. Installing it this way will only make it available to you, not other users that may log into the same Mac. To make the control panel available to all users instead, you could place it in the top level /Library/PreferencePanels folder instead of the one in your home folder.
7. Creating a new share point with the NetContent folder.
  1. Open the System Preferences panel and click the SharePoints icon at the bottom.
  2. Type NetContent into the Share Name field. This is the path the remote render machines will ultimately use to access the content folder on the network.
  3. Set the Group to staff.
  4. Set the Owner & Group both to r/w, but leave Everyone set to none.
  5. Use the Browse button to select the NetContent folder.
  6. Set AppleFileServer(AFS) Sharing to Shared (+).
  7. If you will be using any PC render nodes, also set the Windows(SMB) Sharing to Shared (+).
  8. Click the Create Share button.
8. Creating a new File Sharing only user that can log into this machine from each remote Mac for rendering.
  1. Click the Users & "Public" Shares button.
  2. If any of the current users are selected, deselect them by command-clicking on the selected user in the user list to the left. This will clear out the text fields to the right.
  3. Type NetRender in the Full Name field.
  4. Type netrender in the Short Name field.
  5. Change the Group to Staff. This user's group must be set to the same group that you gave read/write access to when you

created the share point NetContent.

6. Click the Get Next UID button to generate a unique UID number for this user.
7. Leave Public Directory Shared? set to No.
8. Click the Add New User button.
9. When asked to confirm adding a new File Sharing only user, click Yes.
10. Then provide your administrator password when asked.
11. Then SharePoints will ask for a new password for this user. Enter a password twice, and write it down somewhere so you don't forget it.
9. Setting the file permissions for the NetContent folder. You'll need to do this any time you add or change content inside the NetContent folder. All the render nodes need read/write access to this content or the network render will fail.
  1. Select the NetContent folder in the Finder and select File->Get Info.
  2. Set Group to staff.
  3. Set Owner Access & Group Access to Read & Write.
  4. Set Others to No Access.
  5. Click the Apply to enclosed items... button.
  6. Then verify that the change flowed down through all the files by selecting a file inside the NetContent folder and verifying that the permissions are set correctly for that file, using File->Get Info. Depending on where you got the content, you may not be able to change all the permissions this way. In that case use the following commands in the terminal. Substituting the name of your external hard drive or path to NetContent and your username. You'll need to be logged in as an administrator and also supply your administrator's password when you run the sudo command.

```
cd /Volumes/WorkHD
```

```
sudo chown -R ownerusername:staff NetContent
```

```
sudo chmod -R 770 NetContent
```



## Mounting the Content Folder from the Render Machines

Once the NetContent share point is set up. It's quite easy to mount it from each remote Mac. Simply follow these steps on each remote rendering machine:

1. In any Finder window, click the Network icon.
2. Locate and select the host machine on the network, it may show up directly or you may need to open a subgroup depending upon how your network is configured.
3. Once the host machine is selected, click the Connect button.
4. When the log-in screen appears click the Registered User button. Guest access won't work in this instance since you wouldn't have sufficient read/write privileges.
5. Enter the username and password of the NetRender user you created on the host machine using SharePoints. Click the Connect button when ready.
6. You'll now get a dialog box showing all available home directories as well as any share points on the host computer. Select the NetContent share point. Click OK and the NetContent share point will be mounted on the desktop of the remote rendering node.

## Step-by-step LWSN Mac OS X

### Advanced Network Rendering Example

#### Creating a New set of Config Files for ScreamerNet

Before setting up the individual remote rendering machines, we'll first set up a new SNConfigs folder with a master set of LWSN config files independent of your normal LightWave config files. This way you can change your LightWave configs, without affecting the render farm and vice versa. This new config set may then be modified for each remote rendering machine.

For more detailed information see Mastering LightWave's All Important Config Files.

1. Inside the Applications:LightWave10 folder, on the host Mac (the one that will run the controller that will control the remote render Macs), create a new folder named SNConfigs.
2. Duplicate the LightWave cmdLine file as a backup. It's typically installed as: Applications:LightWave10:Programs:LightWave cmdLine
3. Open the LightWave cmdLine file in TextEdit.
4. Edit the LightWave cmdLine contents to redirect LightWave to the new SNConfigs folder. Edit it to read as follows:  
`-O -c"HostHD:Applications:LightWave10:SNConfigs"`  
 Change HostHD to the name of your Mac's main hard drive. See Mastering LightWave's Mac OS X cmdLine Files for more information.
5. Save the LightWave cmdLine file making sure that no ".txt" extension gets added to the file name, or LightWave will fail to recognize it.
6. Launch LightWave and open the Edit Plug-ins panel with Utilities->Plug-ins->Edit Plug-ins...
7. Click Scan Directory, & Choose: Applications:LightWave10:Plugins
8. Close the Edit Plug-ins panel and quit LightWave, which will write out the new set of config files in the new SNConfigs folder.



## Setting up the Host Config File

We'll now verify the LWSN specific settings in the new config file.

See Config Settings for LWSN for more information.

1. Launch LightWave on the host Mac.
2. Open Render->Options->Render Options and verify that Multithreading is 1 Thread.  
See Multithreading for more information.
3. Open the Camera Properties panel and verify that Segment Memory Limit is 32, or higher if you intend to render at high resolutions.  
See Default Segment Memory for more information.
4. Click the yes button when asked if this should be the new default setting.
5. Quit LightWave to save the changes to the config file.
6. You may now switch your LightWave cmdLine file back to what it was before, using the duplicate backup cmdLine file you made. This way LightWave itself will use its own original set of configs, independent of these new SNConfigs. Just keep in mind that if you subsequently add plug-ins or otherwise change your normal LightWave configs, you'll also need to update your SNConfigs if you wish the render farm to also use the new plug-ins or other changes.

## Setting up the Config Files on the First Remote Render Node

See Mastering LightWave's All Important Config Files for more information.

1. On the first remote rendering Mac, create a new Applications:LightWave10 folder.
2. Copy the following folders from the host Mac's LightWave10 folder to the remote Mac's new LightWave10 folder.
  - o Plugins
  - o Programs
  - o SNConfigs
3. Open the Applications:LightWave10:SNConfigs:LightWave Extensions 10 Prefs file on the remote rendering Mac in TextEdit
4. Select the name of your host Mac's hard drive in the first { Entry as seen below.
5. Copy the hard drive name to the clipboard.
6. Select Edit->Find->Find... from the menu.
7. Paste the name of the host hard drive into the Find: field.
8. Type or copy/paste the name of this remote rendering Mac's hard drive into the Replace with: field, and click Replace All. This should replace over 600 or references.
9. Save the LightWave Extensions 10 file making sure that TextEdit doesn't add a ".txt" extension to the file name, or LWSN won't recognize it.

## Setting up LWSN and the cmdLine Files on the First Remote Node

For more information see Mastering LightWave's Mac OS cmdLine Files.

1. Open the directory Render01HD:Applications:LightWave10:Programs and locate the LWSN application.
2. Make two copies (assuming you're on a dual processor Power Macintosh) of LWSN and name them LWSN-01 and LWSN-02, with no file extension.
3. Make one copy of the file LWSN cmdLine and name it LWSN-01 cmdLine, with no file extension.
4. Edit the LWSN-01 cmdLine file's contents to read as follows, all on one line:  
-2 -c"Render01HD:Applications:LightWave10:SNConfigs"  
-d"NetContent" ":Commands:job1" ":Commands:ack1"



Note: you need to specify the path to the config directory on the remote render machine. Don't forget to replace Render01HD with the actual name of your remote render machine's hard drive. We'll be using a remote content directory path, NetContent, which we will share with SharePoints. We'll also be using a command directory, Commands, inside this content directory.

5. Save the cmdLine file, making sure no file extension is added to the filename.
6. Duplicate the LWSN-01 cmdLine file and name it LWSN-02 cmdLine.
7. Edit the LWSN-02 cmdLine file's contents by changing the job and ack number to match the file name's number, as follows:  
-2 -c"Render01HD:Applications:LightWave10:SNConfigs"  
-d"NetContent" ":Commands:job2" ":Commands:ack2"
8. Save the cmdLine file, making sure no file extension is added to the filename.



## Setting up LWSN on the Remaining Render Nodes

Now perform the following steps to set up each additional render node.

1. Copy the Render01HD:Applications:LightWave10 folder from the first remote render node, to each of the others.
2. Open the Applications:LightWave10:SNConfigs:LightWave Extensions 10 Prefs file on each remote rendering node in TextEdit
3. Find and Replace all the instances of Render01HD with the corresponding name of each of the remaining remote render nodes.
4. Save the LightWave Extensions 10 file on each of the remaining remote render nodes, making sure that TextEdit doesn't add a ".txt" extension to the file name.
5. In the Applications:LightWave10:Programs folder on each of the remaining remote render nodes, locate the files LWSN-01, LWSN-01 cmdLine, LWSN-02 and LWSN-02 cmdLine.
6. Change the numbers in these file names with the next available sequential numbers on each of the remaining remote render nodes. For instance. If the next available nodes are also dual processor Macs, you would change all the names from "01" and "02" to "03" and "04". Then the next Mac would get "05" and "06", etc...
7. Edit all the LWSN-## cmdLine file's contents by changing the hard drive name and changing the job and ack number to match the file number in the corresponding file name, as follows:

On the second remote rendering machine:

LWSN-03 cmdLine contents:

```
-2 -c"Render02HD:Applications:LightWave10:SNConfigs"
-d"NetContent" ".:Commands:job3" ".:Commands:ack3"
```

### LWSN-04 cmdLine contents:

```
-2 -c"Render02HD:Applications:LightWave10:SNConfigs"
-d"NetContent" ".:Commands:job4" ".:Commands:ack4"
```

### On the third remote rendering machine:

LWSN-05 cmdLine contents:

```
-2 -c"Render03HD:Applications:LightWave10:SNConfigs"
-d"NetContent" ".:Commands:job5" ".:Commands:ack5"
```

## LWSN-06 cmdLine contents:

```
-2 -c"Render03HD:Applications:LightWave10:SNConfigs"
-d"NetContent" ".:Commands:job6" ".:Commands:ack6"
```

8. If you also wish to render on the host machine, and if it's also a dual processor Mac, duplicate the LWSN and LWSN cmdLine files twice and name them LWSN-07, LWSN-07 cmdLine, LWSN-08 and LWSN-08 cmdLine.
9. Edit all the host machine's LWSN-## cmdLine file's contents by changing the hard drive name and changing the job and ack number to match the file number in the corresponding file name. Just like on the remote nodes. The major difference is that we will also edit the content directory path to use a full local path, since these LWSN instances will access the content folder locally, rather than over the network. In this example the content folder is located on an external drive, named WorkHD, attached to the host Mac. If yours is located elsewhere, use the full path to its location in your setup.

### On the host machine:

#### LWSN-07 cmdLine contents:

```
-2 -c"HostHD:Applications:LightWave10:SNConfigs"
-d"WorkHD:NetContent" ".:Commands:job7" ".:Commands:ack7"
```

#### LWSN-08 cmdLine contents:

```
-2 -c"HostHD:Applications:LightWave10:SNConfigs"
-d"WorkHD:NetContent" ".:Commands:job8" ".:Commands:ack8"
```





## Setting up the Scenes for Batch/Network Rendering

The main difference between setting up the scenes here for advanced network rendering, rather than the previous example in basic network rendering, is that we need to copy (or move) all the content into the network accessible NetContent share point. If you have not already set up your NetContent folder with SharePoints, see Sharing the Content Folder with SharePoints.

For more information about general scene prep, see Successful Scene File Preparation for LWSN.

### 1. Copying all the content to the NetContent folder.

1. Locate the HostHD:Applications:LightWave10:Content:Classic Content folder. This is where the sample scenes are that we will use as an example.

2. Copy the following folders from the Classic Content folder to the NetContent folder that you set up previously with SharePoints. Be sure to maintain the same folder structure.

NetContent:Images:Aviation

NetContent:Images:Reflections

NetContent:Objects:Aviation

NetContent:Scenes:Aviation

3. Select the NetContent folder in the Finder and select File->Get Info.

4. Set Group to staff, Owner Access & Group Access to Read & Write.

5. Set Others to No Access and click the Apply to enclosed items... button.

### 2. Launch LightWave.

3. Use Edit->Set Content Directory to set the content directory to the NetContent folder.

4. Use Items->Load->Scene to load: `“.Scenes:Aviation:MustangLowRes.lws”`

1. Open Render->Options->Render Options

2. Verify the following render range fields:

☐ Render First Frame: 1

☐ Render Last Frame: 60

☐ Render Frame Step: 1

3. Click the Output Files tab.

4. Click the RGB Files button.

5. Type MustangLowRes in the Save As: field.

6. Navigate up one level out of the Images directory to the NetContent folder.

7. Navigate into the folder named Renders.

8. Create a New Folder inside Renders, named MustangLowRes.

9. Click the Save button to dismiss the dialog and accept the changes.

10. Set the RGB Type pop-up menu to LW\_TGA24(.tga)

11. The RGB readout should now read MustangLowRes0001.tga, ...

12. Save the scene.

5. Use Items->Load->Scene to load: `“.Scenes:Aviation:RC_Helicopter.lws”`

1. Open Render->Options->Render Options

2. Verify the following render range fields:

☐ Render First Frame: 1

☐ Render Last Frame: 60

☐ Render Frame Step: 1

3. Click the Output Files tab.

4. Click the RGB Files button.

5. Type RC\_Helicopter in the Save As: field.

6. Navigate to the Renders folder inside the NetContent folder.

7. Create a New Folder inside Renders, named RC\_Helicopter.

8. Click the Save button to dismiss the dialog and accept the changes.

9. Set the RGB Type pop-up menu to LW\_TGA24(.tga)

10. The RGB readout should now read RC\_Helicopter0001.tga, ...

11. Save the scene.

### 6. Quit LightWave.

## Launching the LWSN Nodes on the Rendering Machines

Perform the following steps on each of the remote rendering machines.

For additional information review Sharing the Content Folder with SharePoints& Mounting the Content Folder from the Render Machines.

1. In any Finder window, click the Network icon.

2. Log into the host Mac with the username and password of the NetRender user you created on the host machine using SharePoints. Click the Connect button when ready.

3. Mount the NetContent share point, which will then appear on the desktop.

4. Open the Applications:LightWave9:Programs folder, on the remote rendering machine, not on the host Mac.

5. Launch the next two available LWSN-0# nodes, one at a time, on each dual processor remote machine as follows.

o Remote Render Machine 01: LWSN-01 & LWSN-02

o Remote Render Machine 02: LWSN-03 & LWSN-04

o Remote Render Machine 03: LWSN-05 & LWSN-06

6. Now go to the dual processor host Mac and launch the next two available LWSN-0# nodes, one at a time, as follows.

o Host Machine: LWSN-07 & LWSN-08

7. Note that each LWSN instance will repeatedly write Can't access job file to the window. This is normal, until the render controller is started, which creates the job files.

8. Any of these remote Macs may also be located on the Internet, if the host Mac has a static IP address. In such cases, simply use Mac OS X's Go->Connect to Server on the remote Mac and type in the host Mac's IP address. Mount the NetContent folder and proceed as if it were just another render node, following the previous steps.





---

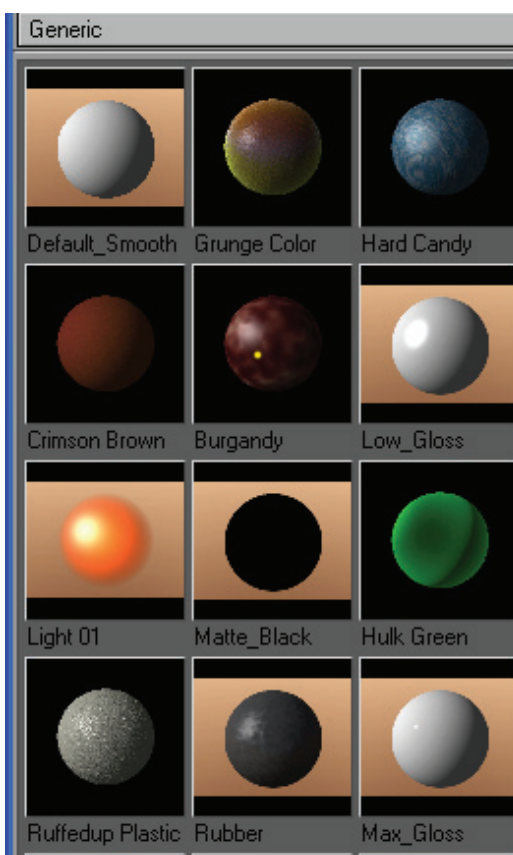
## Chapter 29: Preset Shelf

---



## Preset Shelf

The **Preset Shelf** is a resizable floating window that holds a list of previews along with all of the associated settings in Modeler and Layout. It can be used for settings with surfaces, volumetric lights, **HyperVoxels**, etc. It is accessed using the **Presets** button (**Windows > Presets Panel**) or from some plugin panels. You add to the shelf by double-clicking on the **VIPER** window or preview window, if one exists on the panel (e.g., **Surface Editor**). The shelf survives from session to session.



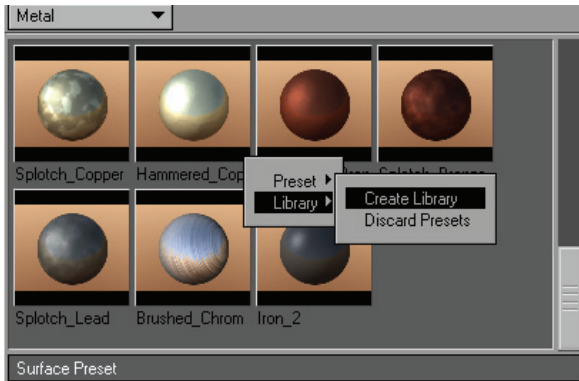
NOTE: Third Party tools can take advantage of the **Preset Shelf** as seen in **Bevel ++** for example.

The window is context sensitive, so if you are working on surfaces, surface presets are displayed, if you are working on



**HyperVoxels**, those presets are shown, and so on. If you have no windows that accommodate Presets, the Presets Shelf will still open, but will appear to be empty.

Each editor has a default library named **Workspace**—a library is a grouping of presets for a particular editor. You can create a custom library by right-clicking over a preview and choosing **Library > Create**. For example, using the **Surface Editor**, you might make a library of Wood presets, a library of Stone, etc. The **Library > Discard Presets** menu clears all of the presets from the displayed library.



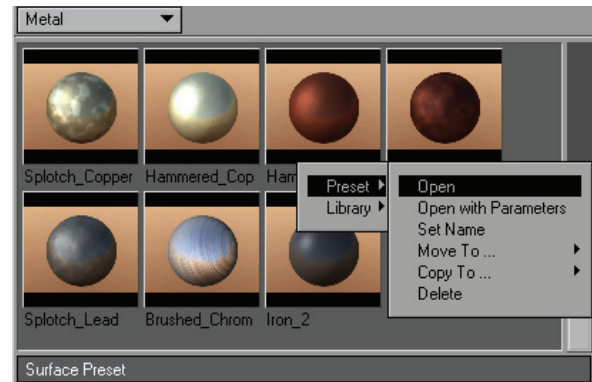
*To apply one of the presets:*

Double-click on a preview or select **Preset > Open** from the **RMB** pop-up menu.

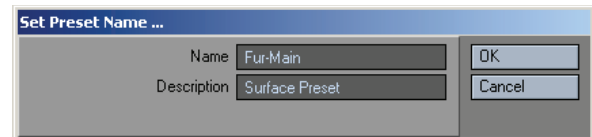
Click the **Yes** button on the confirmation dialog.



NOTE: **Preset > Open w/Parameters** works with certain preset groups, like **HyperVoxels**, and allows you to pick which part of the settings you want to use.



From the **Presets** menu choose **Set Name** to open a dialog box where you can set a **Name** to be used for the sample, it appears underneath the sample image. You can add a longer string of text in the **Description** field, which appears at the bottom of the panel when your mouse pointer is over a preview.



There are also options to **Move/Copy** presets between libraries. The **Delete** option removes the sample from the shelf.



HINT: Keeping your Presets organised will pay off in the end. After a while you may find that you have quite a large collection.





## Chapter 30: Fiber FX



## What is FiberFX?

FiberFX is a new addition to LightWave 3D 9.5 that allows you to create hair and fur effects on your models. Whether it's grass for architectural exteriors, flowing hair on a character, fur on a teddy bear or any other effect which is largely fibrous looking; FiberFX offers you the flexibility to create many different looking styles.



FiberFX Hair Example



FiberFX Grass Example

FiberFX consists of 4 separate plugins, one is located in LightWave 3D Layout, the others in LightWave 3D Modeler.

## Plugin Name Type Location In LightWave 3D

FiberFX Pixel Filter: Layout 'Image Processing' Panel > 'Add Pixel Filter' > 'FiberFilter'

FiberFX Strand Modeler: Modeler 'Setup' Tab > 'FiberFX' > 'FiberFX'

Strand Tool: Modeler 'Setup' Tab > 'FiberFX' > 'Strand Tool'

Strand Maker: Modeler 'Setup' Tab > 'FiberFX' > 'Strand Maker'

Each of these tools falls into one of the two main concepts you need to understand while working with FiberFX; how fibers are created, and how they are rendered.

## How Fibers Are Created

Before fibers can be rendered, fiber guides have to be created first. Within FiberFX there are two ways to create guides, which ultimately allows three approaches to fiber generation. One is to 'grow' them on your model using the 'FiberFX Pixel Filter' in LightWave 3D Layout, the second is to model guide strands using the 'FiberFX Strand Modeler' found in LightWave 3D Modeler, which then control the 'FiberFX Pixel Filter' generated fibers, and the third is to create fully 3D geometry 'fibers' also using the 'FiberFX Strand Modeler'.

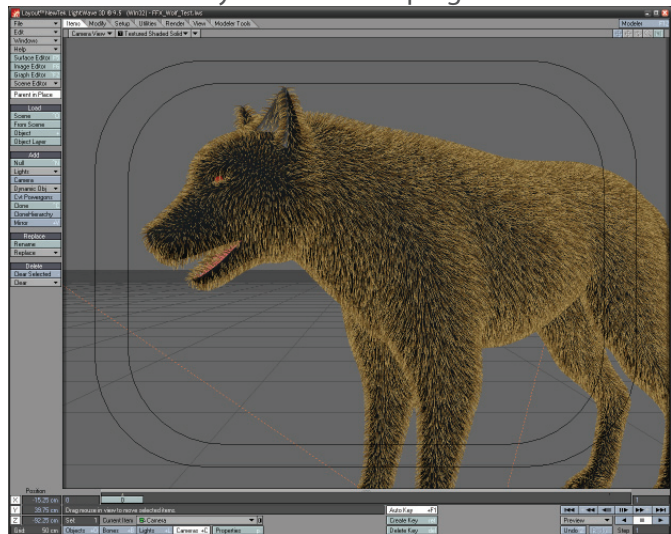
Each of these different fiber creation methods has advantages and disadvantages.





## FiberFX Pixel Filter Generated Fibers

As already mentioned, these are fibers that are 'grown' on your model using the 'FiberFX Pixel Filter' plugin, they require no physical geometry guides as these can be created and styled within the plugin.



FiberFX Pixel Filter Generated Fibers

### Advantages

The main advantage of 'FiberFX Pixel Filter' generated fibers, is that they are much quicker and easier to setup. Within a few clicks you can have fibers on your model. Your models are also smaller in file size as there is no geometry needed to be saved.

The other big advantage are the styling tools available for 'FiberFX Pixel Filter' fibers. They are much more intuitive and interactive to use than their modelled counterparts, although modelled fibers can be controlled much more accurately as they use the LightWave 3D Modeler environment.

Because 'FiberFX Pixel Filter' generated fibers can interact with the volumetric system, instancing of large areas of fibers is possible.

### Disadvantages

As already suggested, the main limitation with 'FiberFX Pixel Filter' fibers is that they cannot themselves be animated using LightWave 3D's dynamics tools. So if you had a character with long hair and you wanted the hair to move naturally, then you would want to use geometry based fibers.

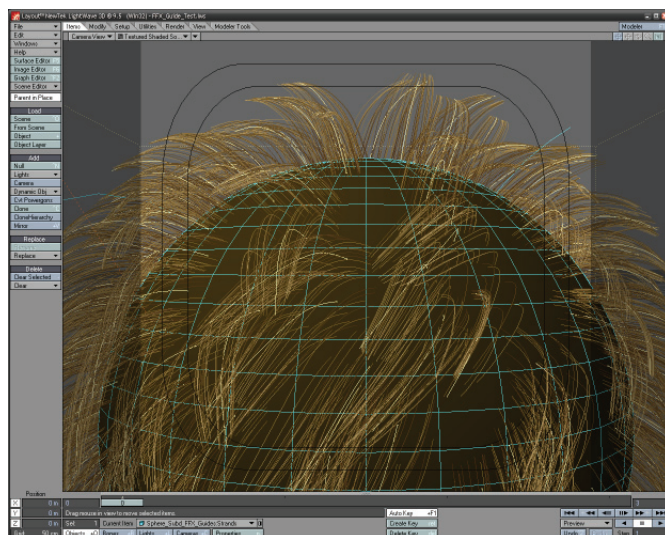
'FiberFX Pixel Filter' generated fibers will move with the underlying mesh however, giving some impression of movement, but not as realistic.

Because 'FiberFX Pixel Filter' generated fibers have no physical 3D volume, they are limited to the finer looking fibers like hair and fur.

## Guide Geometry Based Fibers

Although these fibers are also 'grown' on your model, they differ in that they are added to guide strands created by the 'FiberFX Strand Modeler' and not on your base object.

Strand guides can be used when modelled as two-point polygon chains in 'FiberFX Strand Modeler' which are then used in conjunction with the FiberFX Pixel Filter in LightWave 3D Layout to draw the fibers directly where the 'strands' are located. You can also add more fiber strands around these 'base strands', but there is a limitation which we will go into in the disadvantages section.



FiberFX Pixel Filter Geometry Guided Fibers

### Advantages

Because the generated fibers are 'attached' to geometry guide strands, they can be animated like any other LightWave 3D model using dynamics. As the guide strands are affected by dynamics, the fibers follow them, which allows more natural movement of the fibers.

Guides strands also have the ability to imported and exported to and from other packages, and so can be used for styling hair using other third-party tools such as Worley Labs Sasquatch plugin.

### Disadvantages

Interactive styling of fibers is limited to basic parameters in the 'Geometry' tab in the 'FiberFX Pixel Filter' interface, although it is possible to style the modelled guide strands in 'FiberFX Strand Modeler'.

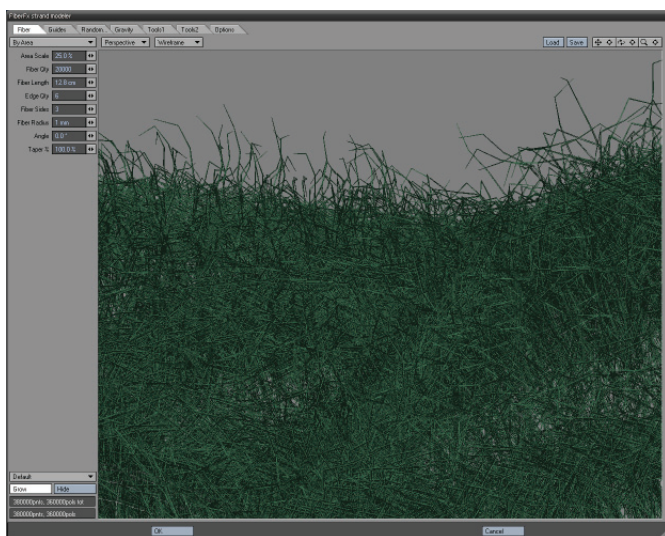
If you want to create human hair using a combination of guide strands and 'FiberFX Pixel Filter' generated fibers, you will need many guides to ensure good coverage of the head surface, as using the 'Cluster' setting (see page XX) to create additional fibers around the guide strands will result in fibers 'leaving' the head surface if the 'Cluster Radius' setting is too large.



## 3D Strand Modeler Geometry 'Fibers'

These 'fibers' as their name suggests, are actual hard geometry, and are created in LightWave 3D Modeler using the 'FiberFX Strand Modeler'.

Geometry fibers can also be given multiple sides so that they have physical 3D volume, at this point you don't need to use the FiberFX Pixel Filter to render them as they will be seen by the LightWave 3D renderer like any normal geometry. They can even be made large enough so that you can create flat areas in which to map images onto. When fibers are created this way, the Strand Modeler also generates UV maps of the polygons ready to be textured, and corresponding weight maps which could then be used to animate the polygons even further. Using this method it's possible to create leaves using a texture map along with a transparency map, or even feathers.



FiberFX Strand Modeler 3D 'Fibers'

### Advantages

Just like guide based geometry fibers, 3D polygonal 'fibers' can be also be animated using dynamics.

Styling of fibers is much more precise as you can use any of Modelers tools to tweak your fibers once created, or even convert existing geometry into FiberFX ready strands and guides.

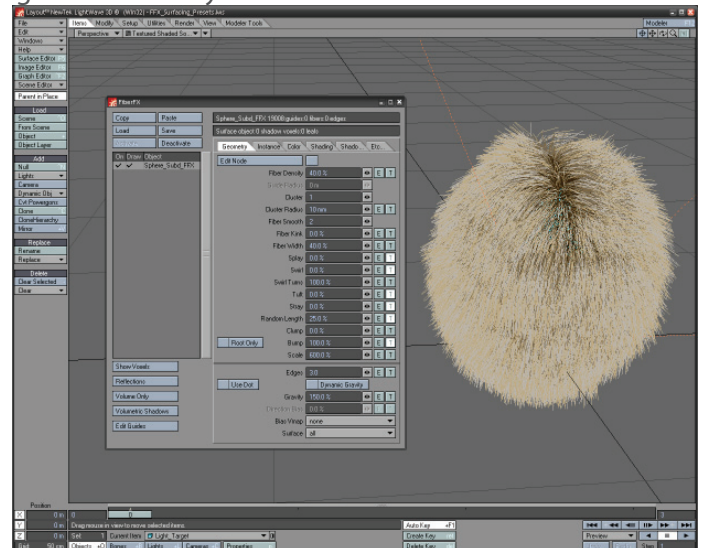
### Disadvantages

The biggest draw back is the heavy geometry that will be created, especially if you have large areas of dense fibers, so doing large areas of grass using this method is not really the best option both for memory, performance and file size reasons.

Setup time is usually longer using this method as there is the extra process of modelling the fibers ready to be taken into LightWave 3D Layout with your model.

## How Fibers Are Rendered

Once you have created your fibers (or more specifically, fiber guides) they are ready for rendering. Fiber rendering is handled by the 'FiberFX Pixel Filter' in LightWave 3D Layout, yes, the same plugin that can also 'grow' fibers on your models.



The 'FiberFX Pixel Filter' takes either the fiber guides you 'grew' on your model, or the strand guides modelled in LightWave 3D Modeler, and renders the fibers using them. The 'FiberFX Pixel Filter' is also where you can set surfacing attributes for your rendered fibers.

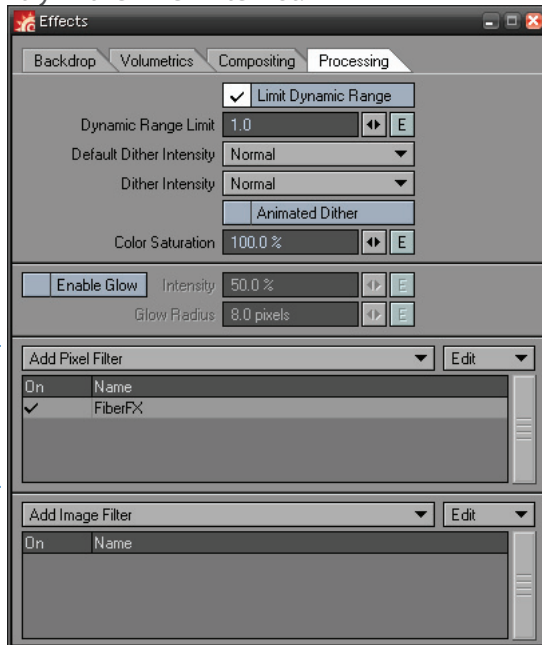
All fiber rendering is handled inside LightWave 3D Layout by the FiberFX Pixel Filter plugin, this is the main 'engine' of FiberFX. For the most part, fiber rendering is a 'post-process' effect, which means once LightWave 3D has finished it's part of the rendering, the FiberFX Pixel Filter plugin takes over and renders the fibers onto the image, in reality FiberFX is working alongside the LightWave 3D render engine which we'll talk about more later, but the actual fibers are drawn to the image at the end, hence the name post-process.



## Fiber Filter Pixel Filter > Introduction

In order to use the 'FiberFX Pixel Filter' it needs to be added as a 'Pixel Filter' in the 'Processing' tab in the 'Effects' window. The 'Effects' window can be found under the 'Windows' popup menu on the main interface, or by pressing the hotkey CTRL F8 (Option F8 for Macintosh users). To bring up the 'FiberFX Pixel Filter' interface once added, double-click on the 'FiberFX' entry in the Pixel Filter list.

Add  
FiberFX  
as a  
Pixel  
Filter



Effects Panel > Processing Tab

## Fiber Filter Pixel Filter > Adding Fibers to a Model

To quickly add fibers a model, select the object's name in the 'Object List' on the left, then click the 'Activate' button, a 'tick' will appear in the 'On' column to indicate fibers are now grown on the model.

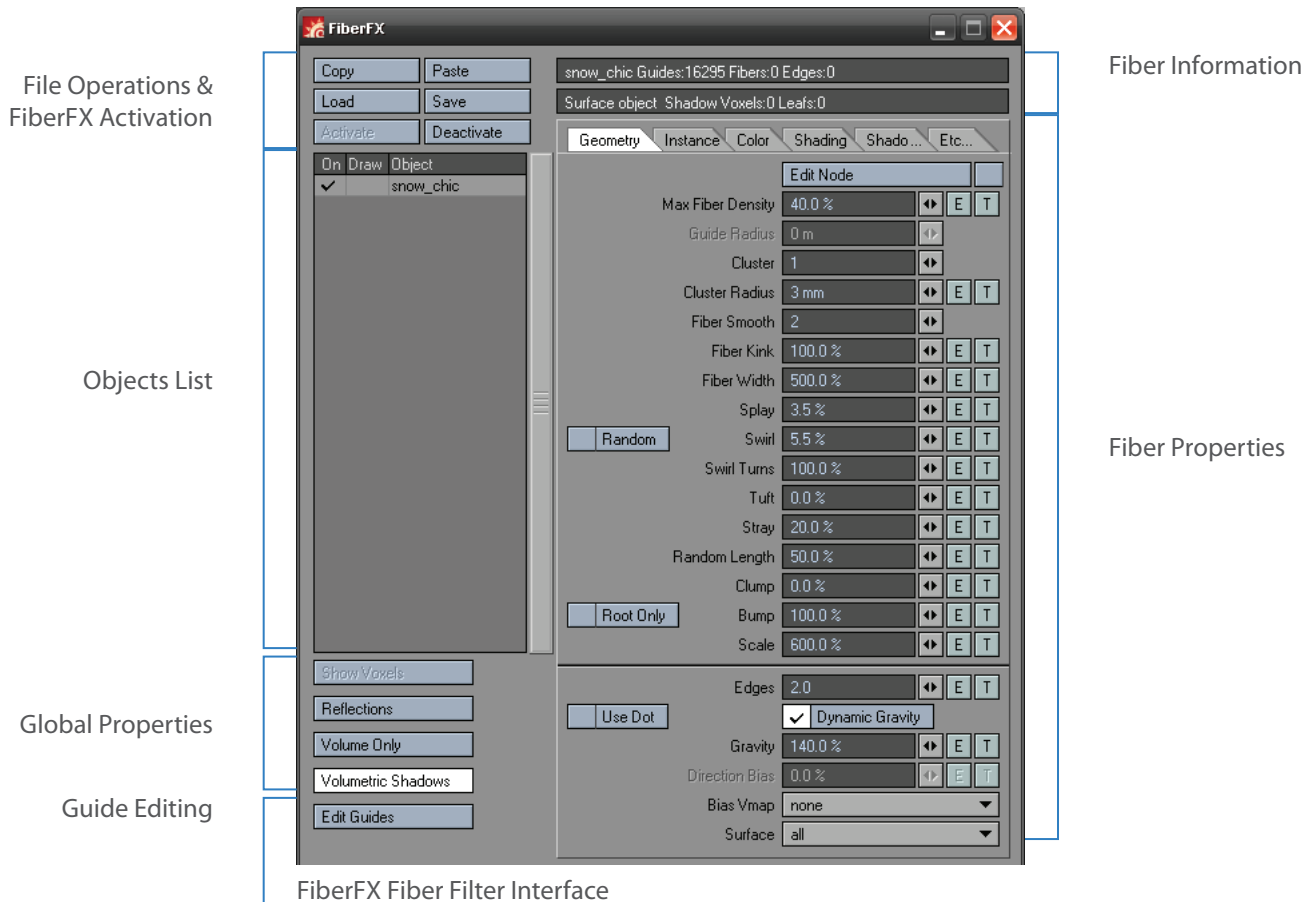
By default 'Draw Fibers' is turned off, to see fibers on your model in the OpenGL view click the blank space in the 'Draw' column next to your object name. You should now see fibers drawn on your model using the default settings. You can set the limit of how many fiber edges are drawn in the OpenGL view by changing the 'Scene Edge Limit' found under the 'Etc' tab. Increasing this number will allow more fibers to be visible in your OpenGL view at the expense of working speed within LightWave 3D Layout.

Once you have added fibers to your model, you can now start to modify how they will look when rendered. In the next part we will go through the 'FiberFX Pixel Filter' interface and the various options available.

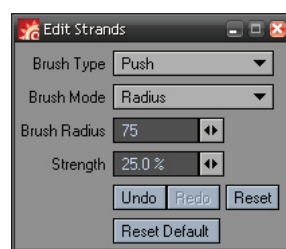


## Fiber Filter Pixel Filter > User Interface Overview

The 'Fiber Filter' panel in Layout (found in the 'Image Processing' Panel > 'Add Pixel Filter' > 'FiberFilter') is for creating 'pixel generated' fibers. Unlike the 'FiberFX Strand Modeler' found in LightWave Modeler, no geometry is actually created, it is a post-process effect. Although it's main usage is for shorter fur type effects, it can also be used to create hair, grass, carpets etc.



FiberFX Fiber Filter Interface



Edit Guides Interface





## Fiber Filter Pixel Filter > Global Controls

There are a number of controls on the main FiberFX interface that allow control over certain global parameters, as well as some other operations.

### Copy / Paste

When you have more than one object with FiberFX applied, you can copy and paste the settings using these buttons.

### Load / Save

Allows you to load and save your FiberFX preset settings to disk. Note any fiber guide styling done using the 'Edit Guides' option is NOT saved in the preset, this information is saved into the scene file.

### Activate / Deactivate

Options to turn on / off fiber generation on selected objects.

### Show Voxels

FiberFX uses LightWave's Volumetric system for creating shadows by creating 'Voxel' objects along the fiber length. Each voxel contains the fibers opacity at that point, which means unlike standard LightWave 3D 'Shadow Maps', the shadows from fibers can be opaque or solid where the fiber density changes. Shadow Maps also require a Spotlight, whereas FiberFX can use any LightWave light, and only needs to create one shadow structure for all lights in the scene, whereas Shadow Maps require this process to be completed for each light using shadow maps.

Clicking this option allows you to see shadow's voxel objects in the main viewport.

### Reflections

A global switch to turn on / off reflection of fibers seen in other objects in your scene (fibers do not reflect within themselves). Currently only the base color of an object can be seen in reflections.

### Volume Only

A switch to draw Fibers using LightWave's Volumetric system instead of using the default post process pixel filter mode. Volumetric fibers are coarser in appearance, but are able to participate in all volumetric effects including volumetric lighting.

### Edit Guides

Brings up the fiber guide styling tools, which allow you to comb fiber guides directly in the main viewport using airbrush tools. This is covered in more detail later in the manual.

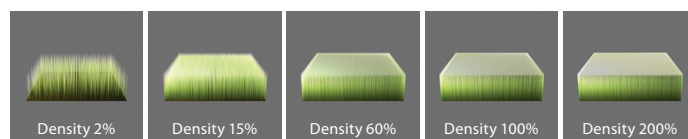
## Fiber Filter Pixel Filter > Geometry Tab

### Max Fiber Density / Fiber Qty

Controls how dense your fibers look on your surface. This setting behaves differently depending on whether you're using surface generated fibers or guide driven fibers.

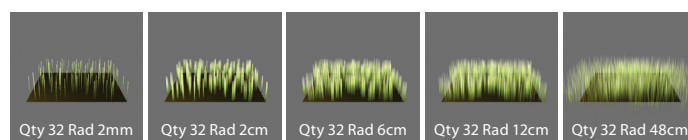
For surface generated fibers 'Max Fiber Density' multiplies the number of fibers by the number of polygons and then distributes them across the entire selected surface area. Small polygons might not receive any fibers if they haven't accumulated enough area for a fiber to be placed.

When using geometry guide driven fibers, the 'Fiber Qty' is the number of fibers generated around each guide fiber, each 'bunch' of fibers form a wisp, the whole wisp will follow the movement or animation of the base guide fiber.



### Guide Radius

When using geometry guide driven fibers, the radius is how far additional fibers are scattered around each guide hair. The limitation is that because fibers are not 'attached' to the surface, increasing the radius too far will force fibers off the surface as can be seen in the examples below. This control is disabled for surface generated fibers as geometry guides are not used.



### Cluster and Cluster Radius

The Cluster setting allows you to add additional fibers around each guide hair, the distance these additional fibers are away from the guide hair is controlled by the Cluster Radius.



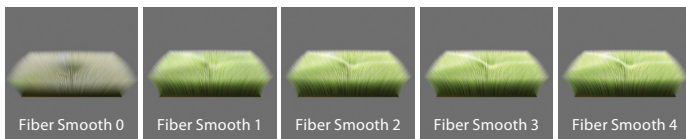


## Fiber Filter Pixel Filter > Geometry Tab

### Fiber Smooth

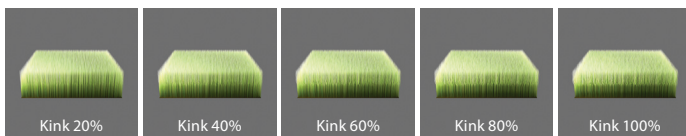
FiberFX draws surface generated fibers as a series of two-point polygon edges, they are not perfectly smooth splines. When the number of 'Edges' setting is greater than 1, you can then use the 'Fiber Smooth' setting to subdivide each edge even further, it works much the same way as 'Subdivision Levels' do on regular LightWave models.

'Fiber Smooth' has four levels of subdivision: Level 1=2 Edges, 2=4 Edges, 3=8 Edges, 4=16 Edges, these edge values are how many times EACH edge in the 'Edges' setting is divided up into. So 3 'Edges' with 'Fiber Smooth' level 2 =  $3 \times 4$  (16) total edges / fiber.



### Fiber Kink

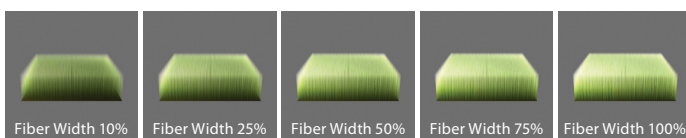
This 'Fiber Kink' setting is only available when 'Fiber Smooth' is set to 1 or above. As fiber edges are subdivided, each newly created edge end point can be perturbed. This creates a kinked look to the fibers. Because subdivision of fibers is performed just before the edges are drawn, they are not used in creation of the shadows, as these are created prior to subdivision. As a result, the 'kinking' effect will not be visible in shadows.



### Fiber Width

Controls how thick or thin fibers appear. Thinner fibers are more transparent and build up density slower, whereas thick fibers are less transparent and build up density faster. Model scale has an effect on fiber width, as does the distance to camera.

Fibers further away are smaller in screen space and are more transparent. The percentage is based on human hair which has been measured at 0.017mm to 0.181mm with blond hair being the thinnest and black hair the thickest. At 100% width a fiber is calculated using a median value of 0.05mm.



## Fiber Filter Pixel Filter > Geometry Tab

### Splay

Controls how much fibers can be tilted away from each other. The splay center is calculated from the centre of each polygon. When using this setting with geometry guide driven fibers, the splay center is the position of the each guide fiber, all the other fibers tilt away from it.



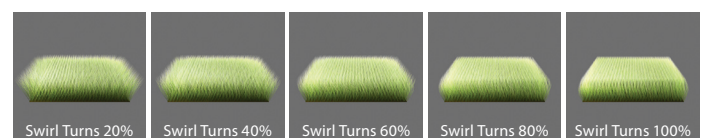
### Swirl

Controls how much fibers can be rotated around each other. The splay center is calculated from the centre of each polygon. When using this setting with geometry guide driven fibers, the swirl center is the position of the each guide fiber, all the other fibers rotate around it. ('Swirl Turns' at 50%). The Random checkbox makes a random starting rotation offset.



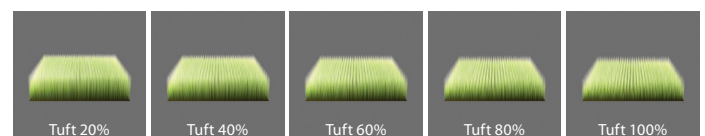
### Swirl Turns

When the 'Swirl' setting is above 0%, the amount of turns can be controlled by the 'Swirl Turns' setting.



### Tuft

Controls how much fibers can be shorten the further away from the guide center they are. The tuft center is calculated from the centre of each polygon. When using this setting with geometry guide driven fibers, the tuft center is the position of the each guide fiber.



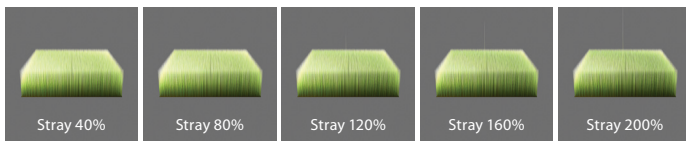




## Fiber Filter Pixel Filter > Geometry Tab

### Stray

One fiber in each wisp can be scaled to a larger size, but following the original shape. This creates a look where just a few strands of hairs poke out from the rest.



### Random Length

Randomizes the length of the fibers to create a less uniform appearance.



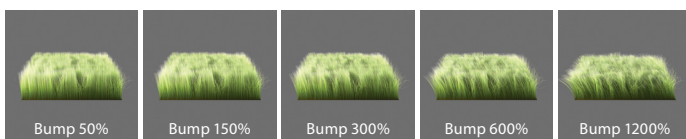
### Clump

Fibers can clump together as if sticky or wet. Only fiber within a wisp can clump together.



### Bump

This effect perturbs the initial fiber direction by simulating a surface bump using the local gradient of the texture. Similar to 'Bump Maps' in the 'Surface Editor' in LightWave. This effect needs something in the texture channel such as a 'Procedural Texture' to be visible.



## Fiber Filter Pixel Filter > Geometry Tab

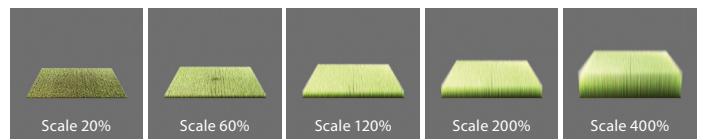
### Bump - Root Only

With the 'Root Only' option switched on, bump is calculated from the root only, giving a different look to how the bump effect is applied.



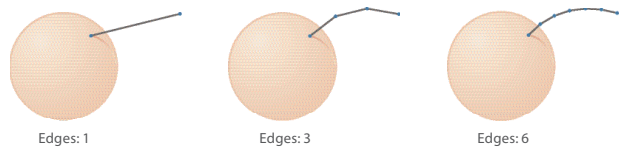
### Scale

Scales all fibers up or down by the specified amount.



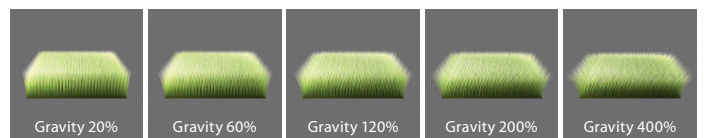
### Edges

The 'FiberFX Pixel Filter' draws fibers as a series of two-point polygon edges, they are not perfectly smooth splines. You can set the number of edges of your fibers (max 127) using the 'Edges' setting. The higher the number, the smoother your fibers will look.



### Gravity

Simulate the effect of gravity on your fibers, pulling them down the more you increase it.

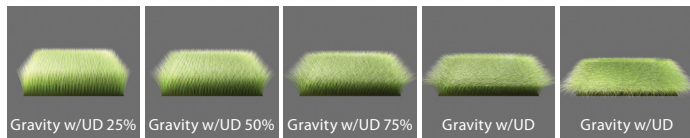




## Fiber Filter Pixel Filter > Geometry Tab

### Gravity - Use Dot

Affects how gravity is calculated. With "Use Dot" inactive fibers on the lowers slope of an object will hang down like gravity is affecting them. With "Use Dot" active the fibers will "hug" the surface and look more like fur.



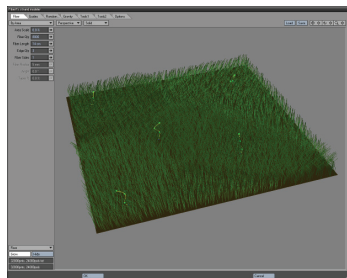
### Dynamic Gravity

If you have applied some gravity to your fibers, when you rotate your model the fibers will also move with the model. By turning on 'Dynamic Gravity' your fibers will always point in the direction of gravity even when the object is rotated.

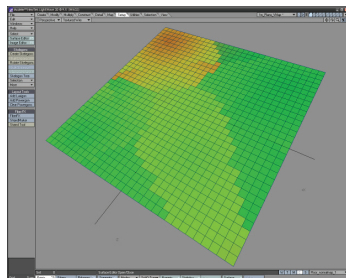
### Bias Vmap

A Bias Vmap is an RGB Vmap (vertex map) which tells the fibers which direction to leave the surface.

In order to create a bias map, you must model your fibers in the 'Strand Modeler' in LightWave 3D Modeler (below left image). The directional information of the fibers is automatically stored as a vmap (below right image) on the model (not the fibers or guides). This is a useful way to control exactly the direction of fur on your models. The vmap isn't created until you build the fibers by pressing the 'Okay' button in the Strand Modeler. If you only intend to use the bias vmap, you can delete any fiber strands or guides that you created, as these are no longer needed for the bias map.



Creating a Bias Vmap in the Strand Modeler



Resulting Bias Vmap After Creation

## Fiber Filter Pixel Filter > Geometry Tab

### Direction Bias

When a Bias Vmap is selected from the popup menu, you can use this control to influence the amount the Bias VMap contributes to the direction of the fibers.



### Surface

This popup gadget allows you to specify which surface is used to grow fibers on. The default is 'All' which will grow fibers on every surface of the selected model, when you select a surface, fibers are grown only on that surface.

If you need to isolate different areas of the model with the same surface name, you can use either weight or image maps (found under the 'T' button to the right of every parameter that supports them) to further control the look of your fibers.



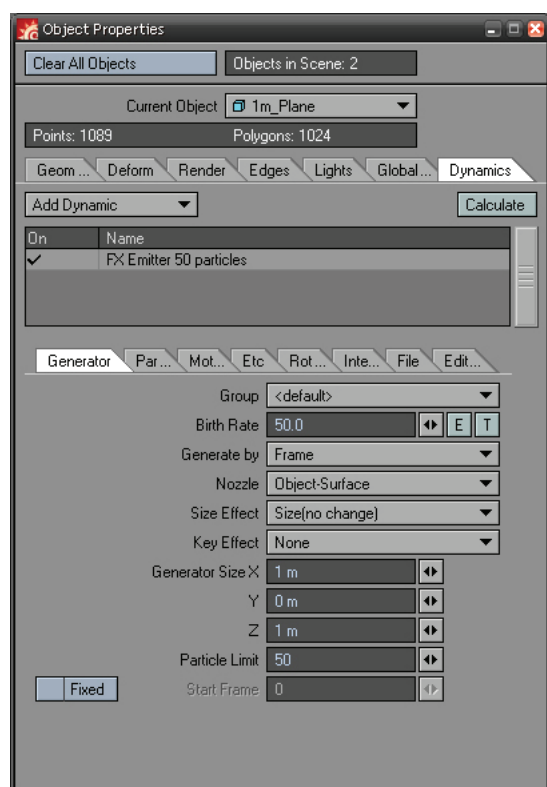
## Fiber Filter Pixel Filter > Instance Tab

### How Instancing Works

FiberFX utilizes LightWave's volumetric system in order to achieve instancing. To activate the instancing controls within FiberFX you must first add a 'Particle Emitter' to your object. This step **MUST** be done **BEFORE** you add FiberFX to your scene.

Instancing also only works with 'Volumetric' fibers, so on the main FiberFX interface, make sure you switch 'Volume Only' on.

To add a particle emitter to you object. First select the object, then bring up the 'properties' panel (keyboard shortcut 'p'). Under the 'Dynamics' tab there is a popup menu called 'Add Dynamic', click that and select 'Emitter' from the menu. Next double click the 'FX Emitter' entry to bring up the properties.



### Adding a Particle Emitter to an Object

When using the particle system for placement of fibers, you must remember that fibers will be subject to the properties and behavior of particles. To avoid having your fibers disappear over time and to ensure they are all visible at the beginning of your animation, set the 'Birth Rate' to the number of 'instances' you require, set the 'Generate By' option to 'Frame', then set the 'Particle Limit' to the same value as the 'Birth rate', these settings can be found under the 'Generator' tab. Next, under the 'Particle' tab, make sure you set the 'Life Time (Frame)' setting to 0, this makes the particles live forever.

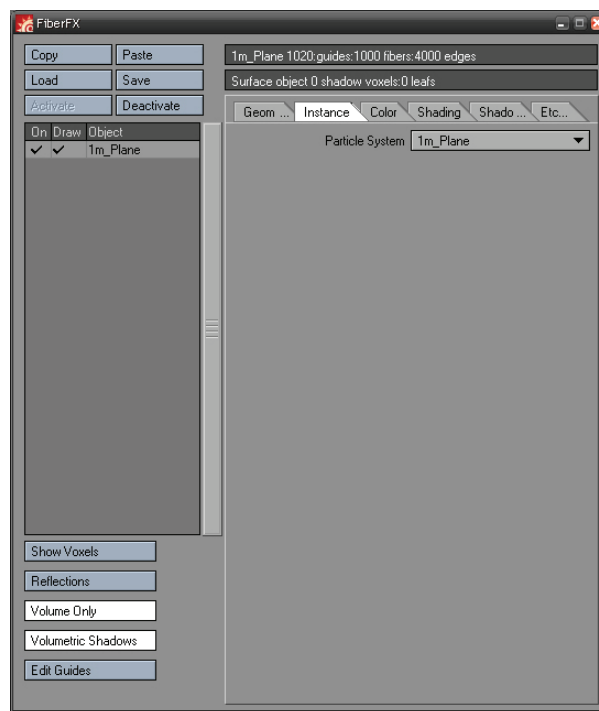
This will set the number of particles to appear all at once at the start of your animation, stop generating (as the particle limit is reached on the first frame) and last until the end of your animation.

Finally, set the 'Nozzle' type on the 'Generator' tab to one that best suits the distribution you're after.

## Fiber Filter Pixel Filter > Instance Tab

### Setting Up Instancing

Once the particles that will drive the fiber instances are setup, you're ready to setup FiberFX to take advantage of them. On the 'Instance' tab, make sure 'Volume Only' is switched on, then select the particle emitter you setup on your object from the 'Particle System' popup menu.



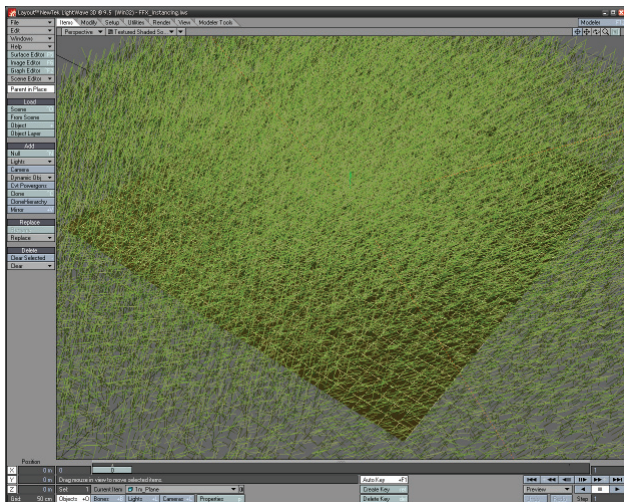
### Activating FiberFX Instancing





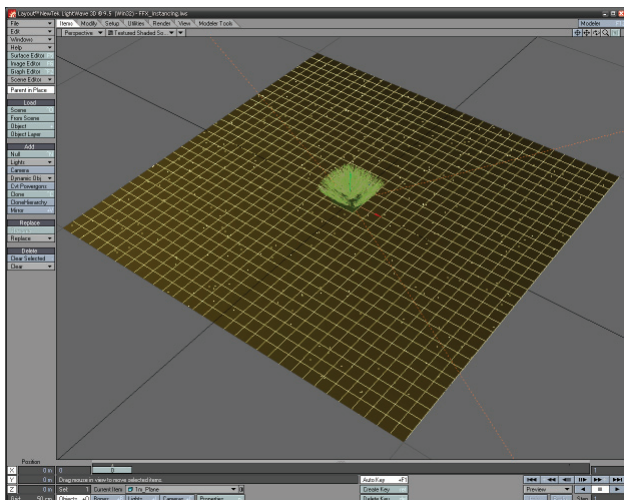
## Fiber Filter Pixel Filter > Instance Tab

Once activated, FiberFX will take whatever fiber setup you have and 'clone' it to each particle location.



FiberFX Instancing Incorrectly Applied

In the example below we want to use instancing to create a number of grass clumps across the ground. If we applied FiberFX to the whole ground surface it would be cloned many times over itself. This is not what we want as can be seen in the image below.

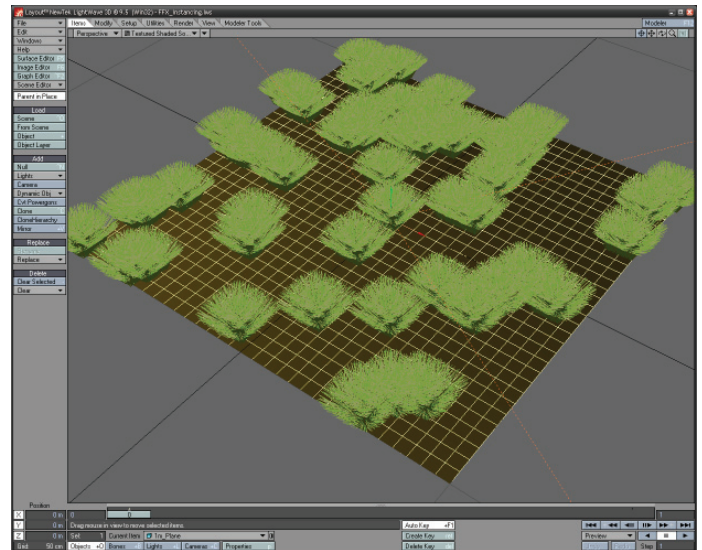


FiberFX Grass Fiber Setup Before Instancing

What we need to do is setup a smaller patch of grass by renaming just a few polygons on the ground plane, we will then use this smaller surface to setup our fibers on. The rest of the ground plane needs to be named differently from the 'grass' surface. In the image below you can see the fibers setup on the surface BEFORE instancing is applied.

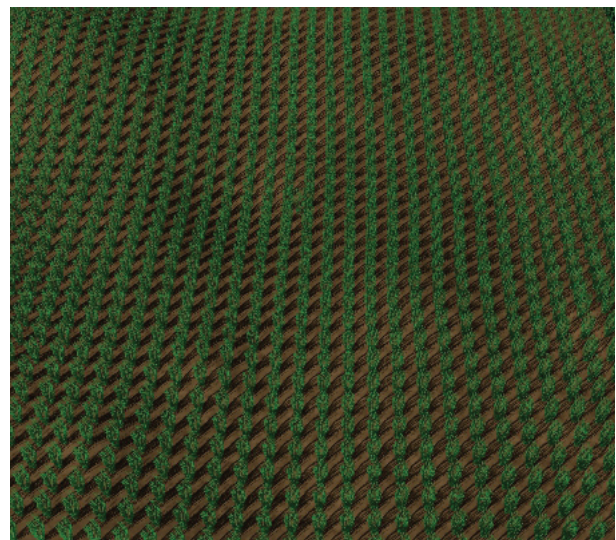
## Fiber Filter Pixel Filter > Instance Tab

Once instancing is activated this is what we now see.



FiberFX Grass Fiber Setup After Instancing

Here are some examples showing what can be achieved using this method.



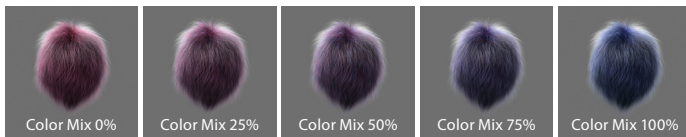


## Fiber Filter Pixel Filter > Color Tab

This tab is where you control the base color of your rendered fibers. There are two main modes for coloring fibers, Mixed (default) and Textured.

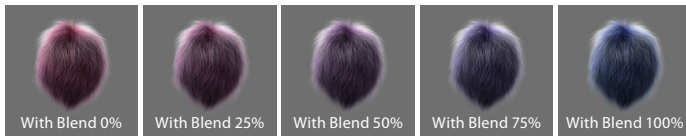
### Color Mode - Mixed

When this mode is selected you can blend between two fiber colors (each with a different root and tip color). You can also set the amount each color influences the overall fiber color. At 0% 'Base / Tip Color 1' will be the predominant color, at 100% 'Base / Tip Color 2' will be the predominant color. In this example we have the first color set to a reddish color, and the second to a blue color.



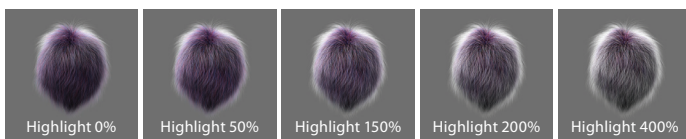
### Blend (Mixed Color Mode)

When 'Blend' is checked, the colors values themselves are actually mixed, as oppose to just the amount of fibers using those colors, you can see this best in the 50% setting below when compared without.



### Highlight (Mixed Color Mode)

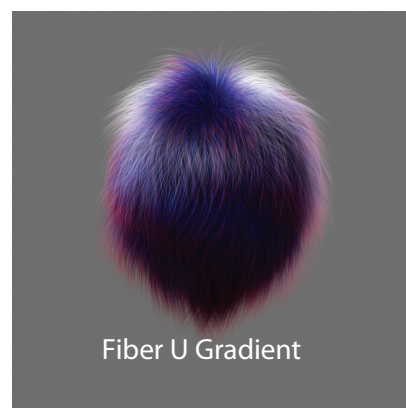
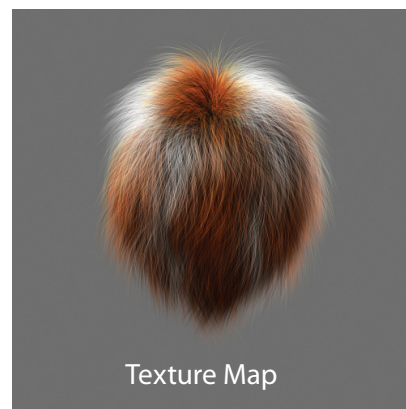
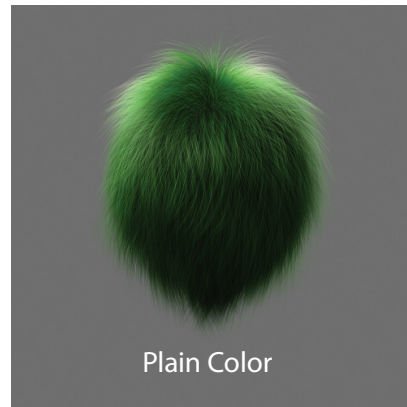
This setting randomizes fiber shading by choosing lighter and darker values based on the original base colors used. The percentage determines how much different the light or dark values are from the original colors.



## Fiber Filter Pixel Filter > Color Tab

### Color Mode - Textured

When this mode is selected you can use either the color picker to select a simple base color. Unlike 'Mixed' color mode, all your fibers will be the same color. However, you can now also use the standard LightWave 3D texture editor to color your fibers, this offers many options for texturing your fibers.

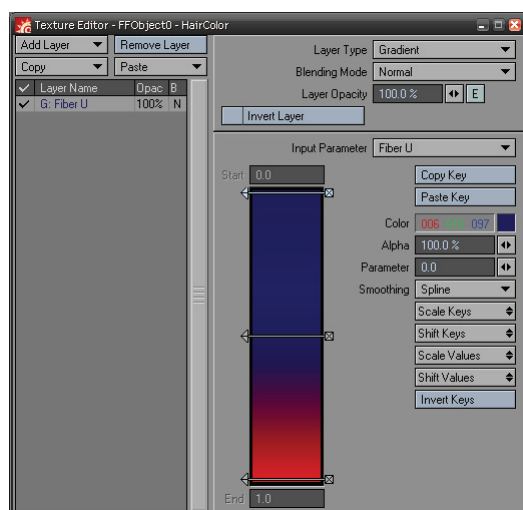
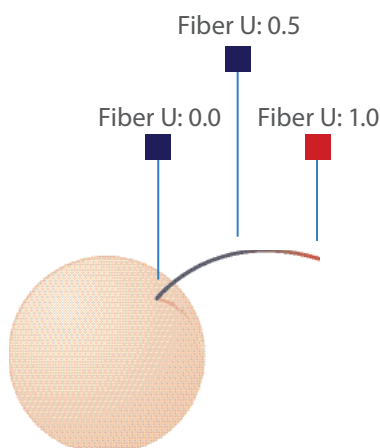






## Fiber U Gradient

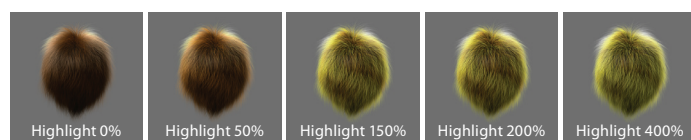
'Fiber U' allows you to control a particular settings value as it travels along the fiber length from root to tip. In the example below we are controlling the fiber colour of some hair from blue at the root to red at the tip. The results of which can be seen in the above right example.



Fiber U Gradient

## Highlight (Textured Color Mode)

The 'Highlight' setting is common to both color modes, and works the same in 'Textured' color mode as it does with 'Mixed' color mode.



## Fiber Filter Pixel Filter > Color Tab

### Color Evaluation

There are two methods for 'Color Evaluation' in the 'Color' tab: 'Interpolated Color' and 'Per Pixel Color'.

### Interpolated Color

Interpolated evaluation is slightly faster than per pixel, but can only evaluate color changes at edge end points, and then blends between them along the fiber.

The limitation of this mode occurs if you want many color changes along a fiber without having enough edge end points for FiberFX to evaluate the color on. So if you had 4 color changes on a gradient along the fiber length, but only 1 edge, you will only see 2 color changes, one change for each end point on the edge.

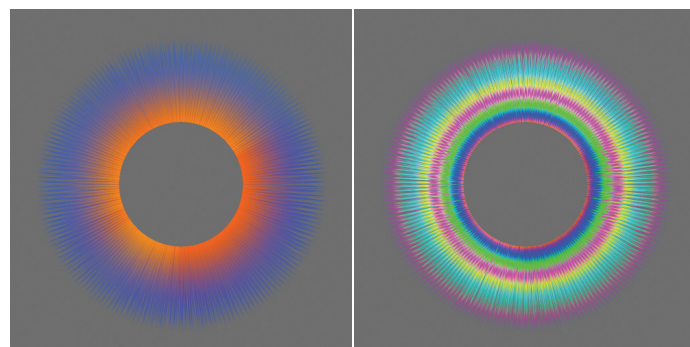
### Per Pixel Color

Although slightly slower than interpolated mode, it can evaluate color changes anywhere along the fiber length, regardless if there is an edge end point there or not.

This means that using 'Per Pixel Color' evaluation would show all the color changes along the length even if you didn't have enough edge end points to 'change colour on'.

The differences are best shown in the examples below. There are 9 color changes along the length of the fiber using a gradient (locations 0, 12.5, 25, 37.5, 50, 62.5, 75, 87.5 and 100%) fiber smoothing is off and edges have been set to 1, which means each fiber has a total of 2 edge end points.

As you can see, the 'Interpolated Color' evaluation only shows two of those color changes along the length of the fiber. Whereas 'Per Pixel Color' shows all 9 color changes.



Interpolated Color:  
1 edge, 2 colors evaluated

Per Pixel Color: 1 edge,  
all 9 colors evaluated



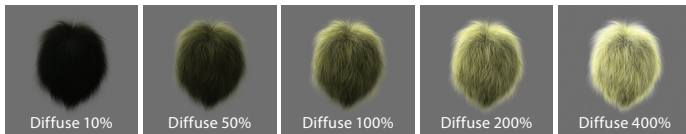


## Fiber Filter Pixel Filter > Shading Tab

This tab contains all the surfacing parameters for your fibers, much of the settings work in the same way they do in the standard LightWave 3D 'Surface Editor'.

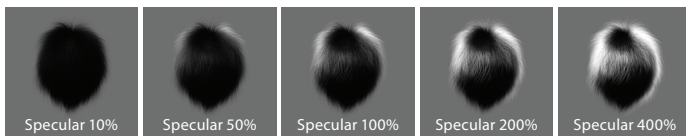
### Diffuse

Determines how much your fibers react to the diffuse setting on lights. Essentially simulating the scattering of light over the fiber surface (not to be confused with sub-surface scattering). The higher the percentage the brighter overall your fibers will look.



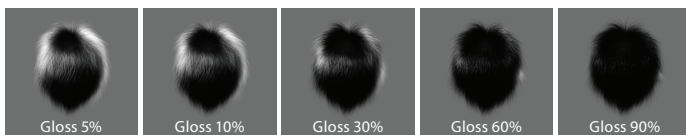
### Specular / Specular Color

Controls how much your fibers will react to specular settings on lights. As with other specularity settings in LightWave this is the 'simulation' of lights being reflected on your fibers. Higher values will increase the intensity of this effect. You can also specify the color of the specular highlights using the 'Specular Color' picker.



### Gloss

In order to see the effect of the gloss setting, you must have 'Specular' above 0%. Much like the gloss setting in the surface editor, the gloss setting controls the spread of the specular highlights on your fibers. This setting works slightly differently however. At 0% gloss is off, once over 0%, low values result in a broader spread of your specular highlights, as the percentage goes higher the spread then becomes narrower. In the examples below, a specular value of 200% was also used.



## Fiber Filter Pixel Filter > Shading Tab

### Luminosity

Another similar acting surface editor setting. Luminosity controls how 'bright' or self-illuminating your fibers look, higher percentages mean brighter fibers. They don't actually emit any light though.

In the examples below 'Diffuse' was set to 75% and 'Specular' 100% to better see the effect.



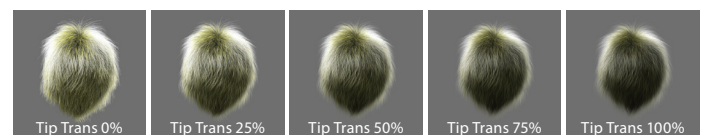
### Ambient

Controls how much ambient light affects your fibers. In order to see any changes, your scene must have 'Ambient Intensity' in your 'Light Properties' panel above 0%. Higher percentages mean your fibers receive more ambient light, and therefore look brighter as a result. In the examples below 'Diffuse' was set to 50%, 'Specular' 100% and 'Ambient Intensity' 50% to better see the effect.



### Tip Transparency

'Tip Transparency' controls at what percentage along your fibers length (starting from the root) your fibers start to have transparency down to 0% at the ends, effectively 'fading them out'. 0% means your fibers have no transparency, and so will appear solid right to the tips. 50% would mean that at 50% along your fibers, they start to 'fade out' until they reach 100% transparent at the tips. Makes hair look thick (0%) or fine and fly-away (100%). In the examples below 'Diffuse' was set to 75% and 'Specular' 100% to better see the effect.





## Fiber Filter Pixel Filter > Shading Tab

### Translucency

Translucency is only really apparent with lights placed behind the subject. Higher translucency settings give the impression of more light passing through the fibers making them lighter. This effects the outer edges of your fibers more than the main body. In the examples below 'Diffuse' was set to 75% and 'Specular' 100% to better see the effect (more noticeable down the right hand side).



### Cuticle Tilt

This setting only works when the 'Specular' setting is above 0% as it controls the position of the specular highlight on the fibers. At 0% the specular highlight is where it naturally lies when light hits it, negative values pull the highlights towards the roots, whereas higher, positive values push the highlights away from the roots and towards the tips. Think of it as a 'manual override' for where specular highlights hit your fibers. In the examples below 'Diffuse' was set to 75%, 'Specular' 100% and 'Gloss' 5% to better see the effect.



### Secondary Gloss / Cuticle Tilt 2

Simulates the secondary 'gloss' or 'shine' fibers display (mainly hair) when light is bounced around inside the fibers as a result of being naturally translucent (not linked to 'Translucency' setting). Works the same way as the first 'Gloss' setting. Cuticle Tilt 2 controls the position of the secondary gloss in the same way 'Cuticle Tilt' controls the first 'Gloss' setting.



## Fiber Filter Pixel Filter > Shadows Tab

This tab contains the controls for tailoring the look of the shadowing of your fibers.

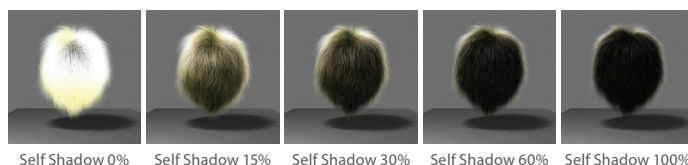
FiberFX uses LightWave's Volumetric system for creating shadows by creating 'Voxel' objects along the fiber length. Each voxel contains the fibers opacity at that point, which means unlike standard LightWave 3D 'Shadow Maps', the shadows from fibers can be opaque or solid where the fiber density changes. Shadow Maps also require a Spotlight, whereas FiberFX can use any LightWave light, and only needs to create one shadow structure for all lights in the scene, whereas Shadow Maps require this process to be completed for each light using shadow maps.

You can view the voxels used to create the volumetric shadow objects in Layout by clicking the "View Voxels" button.

To globally toggle shadows on / off, use the 'Volumetric Shadows' button.

### Self Shadow

Sets the amount of fiber self shadowing. Higher values result in darker shadows, giving the appearance of thicker looking fibers, whereas lower values result in lighter shadows giving the appearance of thinner fibers. In the examples below 'Cast Shadows' was set to 15%. It's also worth noting that the greater your 'Max Fiber Density / Fiber Qty' setting on the 'Geometry' tab, the more dense self shadowing will appear, as there are more fibers to cast shadows.



### Cast Shadow

Sets the density of shadows cast by fibers onto other objects. Higher values result in darker shadows, lower values result in lighter shadows. In the examples below 'Self Shadow' was set to 15%. The shadow you see when 'Cast Shadow' is at 0% is from the object the fibers are attached to, not the fibers themselves.

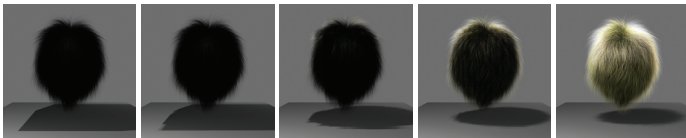




## Fiber Filter Pixel Filter > Shadows Tab

### Shadow Depth

This controls the shadow quality. Shadows are created by subdividing the fiber object into many small voxel boxes. The density of each little box is measured and used to create the shadows. Lower values result in coarser, less accurate shadows, but will render faster. Higher values mean smoother, more accurate shadows, but slower rendering. A value of 16 works best for most setups.



Shadow Depth 2   Shadow Depth 4   Shadow Depth 8   Shadow Depth 12   Shadow Depth 16

### Multisample

Creates multiple shadow samples randomized on a hemisphere about the Light direction. This smooths out shadows at the expense of more shadow rays.

### Shadow Type

There are three shadow types for fibers in FiberFX. 'Interpolated', 'Ray Trace' and 'Point Sample'.



These are the fastest shadows.   Fully traced, but slowest.   Better quality than Interpolated, but slower

### Sample Radius

When 'Shadow Type' is set to 'Point Sample', the 'Sample Radius' option becomes available. The larger the sample radius, the smoother but less accurate point sampled shadows become.



Sample Radius 1   Sample Radius 2   Sample Radius 3   Sample Radius 4

## Fiber Filter Pixel Filter > Etc Tab

The 'Etc' tab contains preferences for the FiberFX plugin, as well as buffer saving options.

### Cull Angle

To speed up the OpenGL display of fibers, this setting eliminates fibers early in the drawing stage that would be behind the object and not be drawn. At 100% fibers beyond the objects profile edge will be culled, at 0% no fibers are culled.

### Fade Angle

This is the angle the fibers fade out over. Tweak this value to have long back facing fibers fade gradually instead of just disappearing from the OpenGL view.

### Scene Edge Limit

Sets the maximum number of fiber edges drawn in the OpenGL view. The higher the number the more fibers will be visible at the expense of slower OpenGL responsiveness.

### World

Turning this option on gathers fiber coordinates in world space and not in relation to the object parent coordinates. Typically you would have this setting off, but if you are using LightWave Dynamics to animate fibers, you need to turn this on and unparent to gather the coordinates correctly.

### Before Volume

Runs the pixel filter and sets the depth buffer before any volumetric plug-ins. This allows volumetric effects like LightWave's HyperVoxels to contribute to the FiberFX depth buffer.

### Save RGBA

Saves the RGBA buffers to a separate file. When this item is active the fibers will not be drawn into the image filter buffer. Make sure you select a file format capable of saving in RGBA (RGB + Alpha) or you will lose the alpha channel.

### Save Z

Saves the depth buffer information in separate floating point buffer. When this is selected, Z buffer values will not be written to the image filter buffer. Make sure you select a file format capable of saving data in floating points.

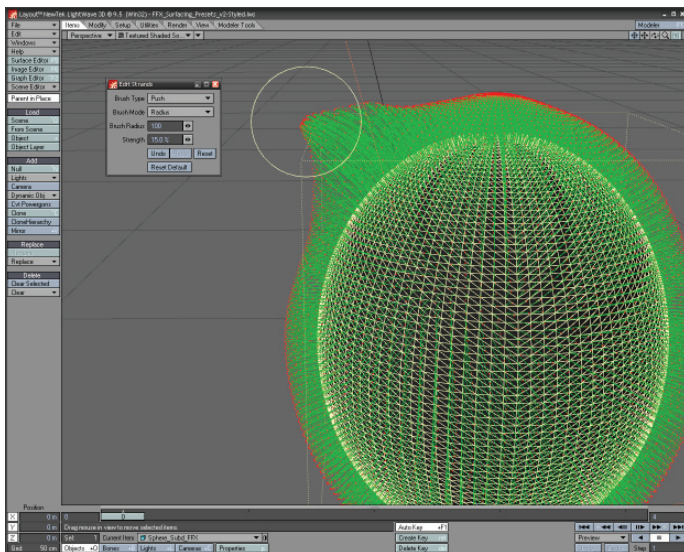


## Fiber Filter Pixel Filter > Editing Strand Guides

FiberFX allows airbrush like styling of fiber guides directly within the LightWave Layout OpenGL viewport. These tools are very intuitive to use, and allow more natural styling of fibers than using just the basic parameters found in the 'Geometry' tab.

To bring up the styling tools click the 'Edit Guides' button in the lower left corner of the main FiberFX interface. Once clicked, the main FiberFX interface will be hidden and you will be presented with 'Edit Strands' dialog box.

The fibers also become hidden and you will see the guides that control them instead, these are what you will be styling. It's worth noting at this point that when you begin styling your guides the 'Edges' setting in the 'Geometry' tab will become locked, so either make sure you have edges set to what you want before styling, or save a 'before styling' version of the scene to preserve the ability to edit edges.



## Fiber Filter Pixel Filter > Editing Strand Guides

### Brush Type - Push

Push mode is similar to combing, guides can be dragged in the direction you move the mouse.

### Brush Type - Scale

Scale mode will increase the length of the guides, if you use the right mouse button while dragging the guides will decrease in length.

### Brush Type - Straighten

This mode will straighten the guides based on the 'normal' direction from where the guide started.

### Brush Type - Single

Single mode allows the 'pushing' of individual guides, useful for tweaking any stray guides that may have occurred as a result of styling guides.

### Brush Mode - Radius

Uses airbrush like manipulation of the 'Brush Types', anything within the brush circle is edited, except on 'Single' where only one guide will be moved.

### Brush Mode - Global

Every guide on the object will be edited equally, allows en-masse editing.

### Brush Mode - Surface Hug

Works similar to 'Radius' mode, but follows the nearest surface on the object.

### Brush Radius

Sets the working size of the brush when 'Brush Type' is set to 'Radius'.

### Brush Strength

Sets the power of the brush, smaller values allow finer tweaks, larger values edit guides much more aggressively. Entering negative values 'invert' the direction of the 'Scale' brush type without needing to use the right mouse button while brushing.

### Undo / Redo

Allows you to undo / redo brushing strokes whilst within the guide editing interface.

### Reset

Resets all changes made going back to when you last opened the 'Edit Guides' interface.

### Reset Default

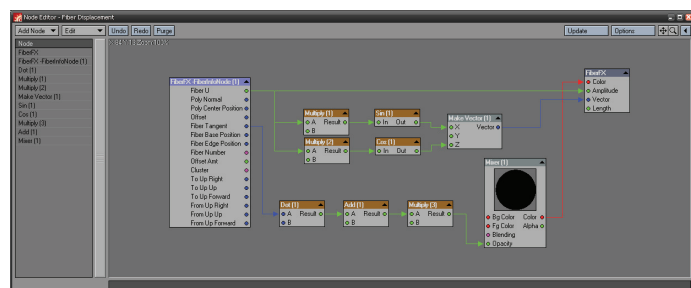
Completely resets ALL styling changes, use with caution, as you cannot undo this change.





## Fiber Filter Pixel Filter > Node Editor

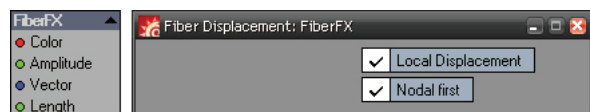
The Node controls for FiberFX are more advanced than the basic parameters found on the 'Geometry' tab, and so, have been deliberately left until the end of this section before introducing them. Having an understanding of the basics in FiberFX will hopefully help making node editor concepts easier.



## Node Editor Usage

The node editor in FiberFX allows you to control the appearance of the fiber 'geometry' styling, as well as the base color of fibers. Once opened you are presented with a standard LightWave 3D Node Editor. The FiberFX 'root node' has 4 inputs that you can control: Vector, Amplitude, Length and Color. When anything is plugged into the 'Color' input, the Node Editor will override that parameter, but the Vector and Amplitude inputs allow blending of any calculations within the Node Editor and any settings changed on the 'Geometry' tab

Double-clicking on the FiberFX root node will display two parameters for how the Node Editor will calculate the resulting direction.



FiberFX Root Node & Properties

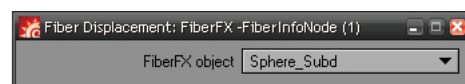
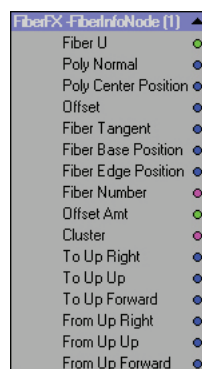
'Local Displacement' orients the displacement to be performed along the underlying polygons normal, this mode is useful for creating swirls, as the ripples radiate down the fiber. Turning it off uses object space displacement, use this mode for things like fields of waving grass.

'Nodal First' changes the order that the nodes and other effects are evaluated, for instance, if you have gravity set to make the fibers droop, it is possible to set the nodes to override the effect of the gravity. The default setting for nodes first will prevent this.

## Fiber Filter Pixel Filter > Node Editor

### FiberInfoNode

The 'FiberInfoNode' allows access to fiber edge information, which you can then use along with regular nodes to modify them, this allows you to create more custom styling of fibers not available by using the standard controls on the 'Geometry' tab on the main FiberFX interface. Double-clicking on the 'FiberInfoNode' allows you to select which FiberFX object to gather the data from.



FiberFX FiberInfoNode & Properties

### Fiber U

The position of the evaluation down the length of the fiber. Starting from 0.0 at the root to 1.0 at the tip. Use this as an input to opacity on a node to control effect strength down the length.

### Poly Normal

The normal vector coordinates of the underlying polygon that the fiber is growing from.

### Poly Center Position

The center position of the underlying polygon that the fiber is growing from.

### Offset

The offset position of the fiber currently being evaluated.

### Fiber Tangent

The normal vector of the edge currently being evaluated.

### Fiber Base Position

Position in local coordinates of the base of each fiber currently being evaluated.

### Fiber Edge Position

Start position in local coordinates of the edge currently being evaluated.

### Fiber Number

Returns the fiber number that is being evaluated



## Fiber Filter Pixel Filter > Node Editor

### Offset Amt (Amount)

The offset of each added fiber from the parent 'wisp' or guide fiber.

### Cluster

Returns the fiber's Cluster number. Use it to determine if a clustered fiber is being evaluated.

### To Up Right

Transform from polygon coordinates to up right.

### To Up Up

Transform from polygon coordinates to up up.

### To Up Forward

Transform from polygon coordinates to up forward.

### From Up Right

Transform from up right to polygon coordinates.

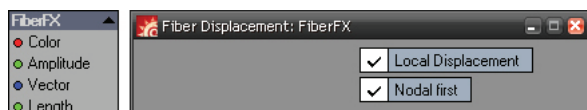
### From Up Up

Transform from up up to polygon coordinates.

### From Up Forward

Transform from up forward to polygon coordinates.

## FiberFX Root Node



FiberFX Root Node & Properties

### Color

Changes the color of the fibers. Use a Fiber U input on a gradient to change colors based on the position along the fiber length.

### Amplitude

Use this input to change the amount of amplitude of the Vector input's perturbation. Without any amplitude input, 100% of the Vector input will be used, often resulting in harsh changes of direction.

### Vector

Controls the direction of the fiber edge vector. The old and new fiber directions are interpolated together by the Amplitude amount.

### Length

Controls the length of the fiber.

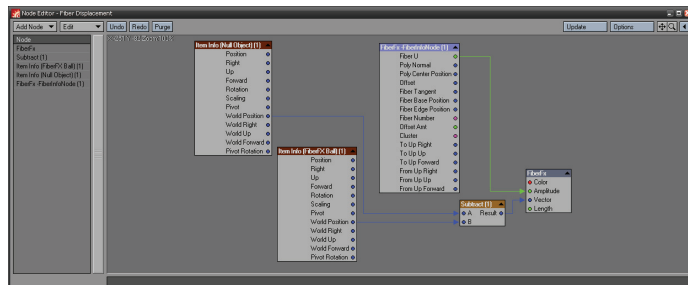
## Fiber Filter Pixel Filter > Node Editor

### Example Node Flow

Here is an example of what can be achieved using the node editor.

### Faked Fiber Dynamics

This very simple node flow gives the illusion of dynamics on the fibers.

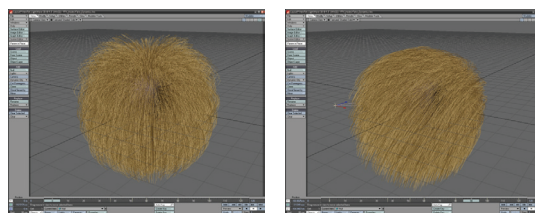


Fake Dynamics Node Flow

The scene has a simple sphere object with fibers grown on the surface and a NULL object.

The fibers vector direction is controlled by subtracting the NULL's world position from the sphere's world position. The amount of the resulting vector direction is then controlled by the Fiber U position, so that at the root of the fiber the strength is low, and higher at the tip, this gives the illusion of 'fading out' of the movement.

In Layout, the sphere and NULL object are hand animated to move the fibers in the general direction they would move in reality to simulate dynamic movement. Clearly this is not as accurate as using actual LightWave Dynamics, but it's a neat solution to animating 'Pixel Generated' fibers that usually can't be animated using LightWave Dynamics.



The result using the node editor to simulate dynamic movement of fibers



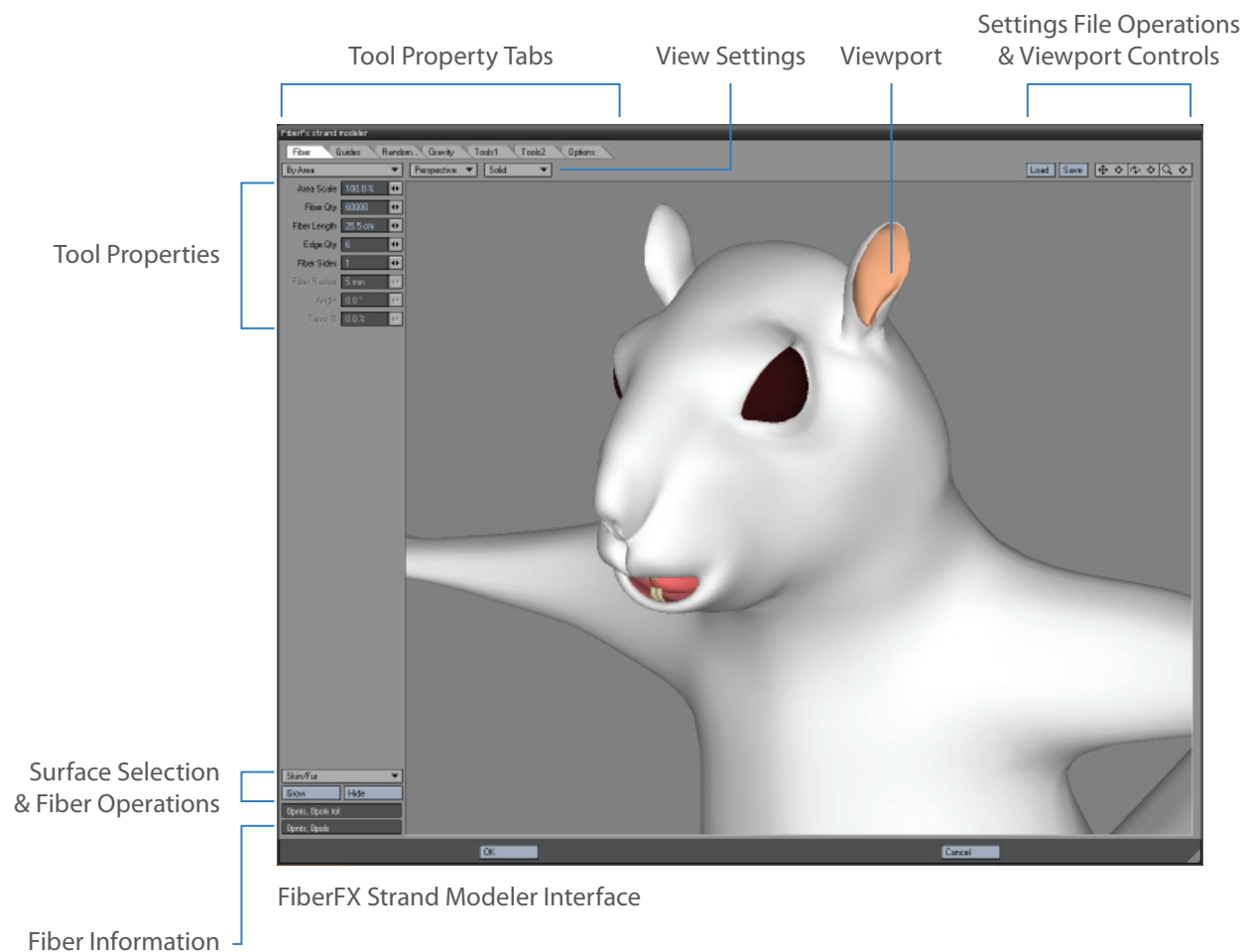


## FiberFX Strand Modeler > Introduction

There are a number of tools available in LightWave 3D Modeler for FiberFX. These tools not only allow you to build 2-point poly chain fiber guides for use with the FiberFX Pixel Filter plugin in Layout, but the creation of full 3D fiber geometry, Vertex Bias maps for controlling fiber direction and auto-creation of UV maps for 3D fibers.

## FiberFX Strand Modeler > Interface Overview

The 'Strand Modeler' is the main part of FiberFX inside LightWave Modeler. It can be found under the 'Setup' tab > 'Fiber FX' title. When opened the Strand Modeler interface looks like this:



FiberFX Strand Modeler Interface

## Viewport Hotkeys

The OpenGL viewport window supports many of the same hotkeys as LightWave 3D. Holding down the Alt key while dragging in the viewport rotates the view, Alt + Ctrl zooms and Alt + Shift pans the view.

The 'a' key auto fits and centers the object in view. The numerical keys 1 - 6 changes the viewport to show front, back, top, bottom, right and left views respectively.



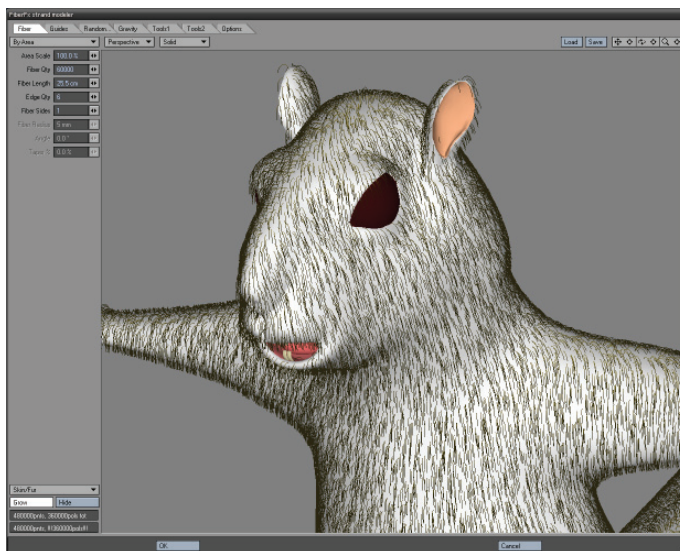
## FiberFX Strand Modeler > Global Controls

### Surface Selection & Controls

In the lower left hand corner, you can select which surface on your model to grow your fibers on using the popup menu. You are not limited to just one surface, selecting a different surface name allows you to grow fibers with different settings on each surface in one fiber modelling session.

Once you have selected your surface, click the 'Grow' button, this will then add fibers to the selected surface using the current settings. Switching between surfaces allows you to edit the fibers parameters for that particular surface.

You can also click the 'Hide' button should you need to clean up the view while working on a particular part of your model.



FiberFX Strand Modeler showing fibers with different

### Load / Save

If you want to save your fiber editing session for later, you can save and load using these controls (you will need to save each surface setting if you have more than one). Note, any guides you've added will not be saved. To store guides you must click the 'Okay' button to complete the session. Fibers will be created along with the guides (each in different layers). To recall your session you must first load the settings, then load the guides using the 'Add Guides' popup menu found under the 'Guides' tab.

## FiberFX Strand Modeler > Fiber Tab

### Fiber Distribution

There are two methods for distributing fibers over your models surface, 'By Area' and '1 Per Polygon'. Both of these methods also have a 'Scaled' option whereby the size of the underlying polygon is taken into consideration, you can adjust this compensation by adjusting the 'Area Scale' property. This gives a total of four actual distribution methods.

#### By Area

The number are fibers in the 'Fiber Qty' setting are scattered across the whole surface randomly.



Fiber Distribution 'By Area'

#### By Area Scaled

The number are fibers in the 'Fiber Qty' setting are scattered across the whole surface randomly but scales the length of the fibers according to the underlying polygon area size. This scaling can be adjusted by the 'Area Scale' control.



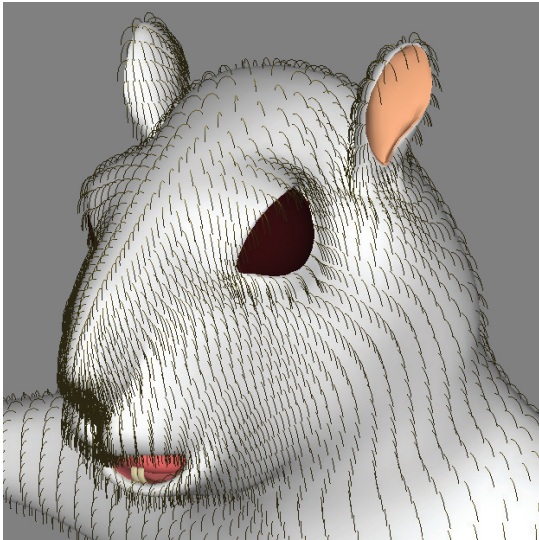
Fiber Distribution 'By Area Scaled'



## FiberFX Strand Modeler > Fiber Tab

### 1 Per Polygon

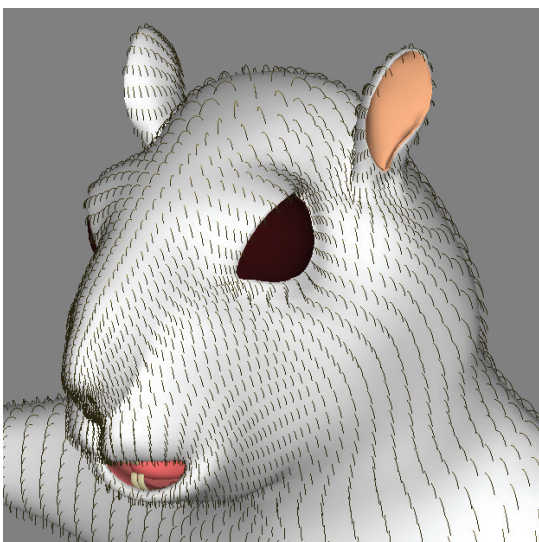
Places one fiber at the center of each polygon on your surface, this creates a very ordered appearance. If you use these fibers as guides in the 'FiberFX Pixel Filter' in LightWave 3D Layout, you can create many more 'virtual' fibers clustered about the centre of these 'guides', giving the appearance of more fibers.



Fiber Distribution '1 Per Polygon'

### 1 Per Polygon Scaled

Places one fiber at the center of each polygon on your surface, but scales the length of the fibers according to the underlying polygon area size. This scaling can be adjusted by the 'Area Scale' control.

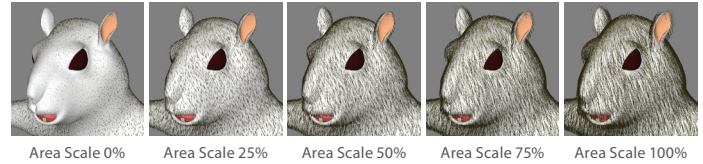


Fiber Distribution '1 Per Polygon Scaled'

## FiberFX Strand Modeler > Fiber Tab

### Area Scale

This setting becomes available when the fiber distribution method is set to 'By Area Scaled' or '1 Per Polygon Scaled'. It adjusts the percentage of scaling from the largest to the smallest polygons.



### Fiber Qty (Quantity)

Determines the number of fibers to be used in 'By Area' distribution mode.

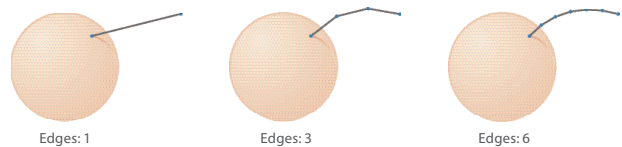
### Fiber Length

Set the length of the fibers. If 'Area Scale' is active this would be the length of the polygons with the largest area. Length is measured from root to tip with the fiber in an absolutely straight position.

Effects such as Curl, Kink, and Gravity change the shape of the fiber, and may cause the overall length to appear shorter.

### Edge Qty (Quantity)

The number of edges on each fiber strand. The more edges the smoother the strand. If you render your fibers using the 'FiberFX Pixel Filter' in LightWave 3D Layout, fiber strands can be further subdivided by using the 'Fiber Smooth' setting. If required, you can cover an object with single point polygons by setting the 'Fiber Sides' setting to 0. The maximum edges per fiber is 127.



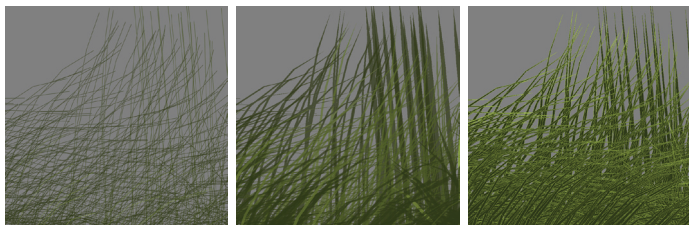




## FiberFX Strand Modeler > Fiber Tab

### Fiber Sides

Fibers can more sides if you intent to use FiberFX for creating fully 3D 'fibers'. 1 side will create 2-point polygon strands which can be used as guide strands. 2 sides creates flat polygon 'blades', and any number above 2 will create full 3D fiber objects.



Sides 1: 2-point poly chains

Sides 2: Flat 2D blades

Sides 3+: Full 3D geometry

### Fiber Radius

The setting controls the thickness of the fiber at the base. It only becomes available when the 'Fiber Sides' setting is 1 or above.

### Angle

Multi-sided fibers can be rotated around their axis using this control.

### Taper %

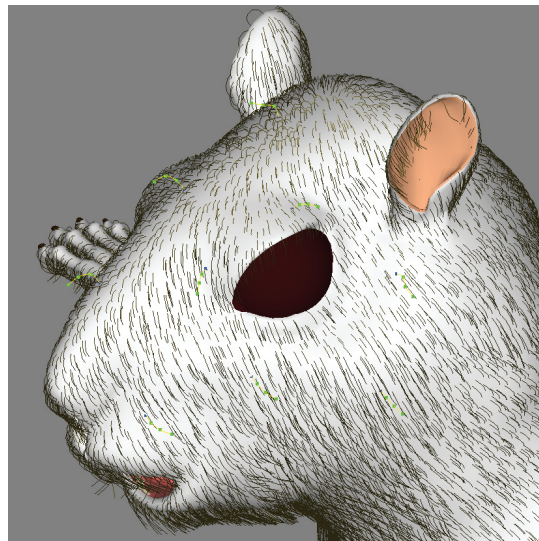
Multi-sided fibers can be set to taper along their length, from root to tip. Higher percentages will create sharper pointier looking fibers. If set to a negative value, fibers can be made to have blunt flat tips. It only becomes available when the 'Fiber Sides' setting is 1 or above.

## FiberFX Strand Modeler > Guides Tab

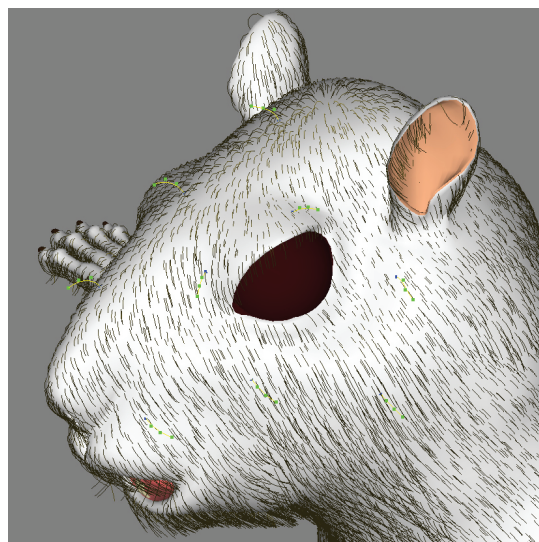
### Spline Type

Changes the spline calculation method for all splines (you can toggle between them).

'Interpolating' splines pass through each knot of the guide. Whereas 'Approximating' splines act more like Bezier splines. 'Interpolating' splines can be bent tighter, 'Approximating' splines are smoother.



'Interpolating' Splines



'Approximating' Splines



## Add Guide

To place a guide on your model to control the fibers, click this button (or press the hotkey 'g') then select a polygon on your model that will serve as the place holder for your guide. You can only place one guide per polygon.

To add many guides in one go, use the 'Grow' button to add fibers to your model and set the 'Fiber Qty' setting to the number of guides you require. Then click the 'Okay' button to commit your settings. Run the Strand Modeler again and use the 'Add Guides' popup menu selecting the layer that contains the fibers you just created. These will then load up as guides and not fibers. Or you can select 'Current Fibers' option which is a much quicker way of doing the same operation.

## Scale (Guides)

Scales the length of the selected guides, this will also scale the length of any fibers they are controlling as a result.

## Del Guide (Delete Guide)

To delete a guide hold down the 'Shift' key and drag a box selection using your mouse over the guide you wish to delete (normal left mouse clicking doesn't select a guide as this is how you move guide knots, not select them). You can continue to use this same method to select multiple guides. When a guide is selected the last knot will be highlighted white. You can now click the 'Del Guide' button to remove the selected guides.

## FiberFX Strand Modeler > Guides Tab

### Add Node (Guide Knot)

To add a new node (or knot) click this button, It will add a node / knot to ALL guides. Currently all guides must have a uniform node / knot count.

### Del Node (Guide Knot)

Clicking this button will delete the last node or knot from ALL guides.

### Add Guides (Load Guides)

You can add or load guides in a number of ways using this popup menu. You can load guides previously saved on layers in your object. 2-point polygon fiber chains can also be loaded back in as guides, and not fibers.

Guides can be created from fibers currently growing on the surface. Note: guides created using this method take on the exact shape of your current fibers, so any styling done to them will be reflected in the shape of the guides.

You can also create guides based every polygon on the currently selected surface - guides will be created matching each polygons 'normal' vector. Or you can create guides based on every single point on the currently selected surface, again these will be created using the point normal vector.

### Save Guides

Saves the current guide set to the next available layer in Modeler.

### Select Node (Knot)

This popup allows you to select either the First / Last node or knot on every guide (it selects all knots on every guide because any deletion of knots must be equal on all guides). You can also use Previous / Next to step through intermediate knots, or use the 'None' option to deselect.

### Clear Guides

Clear all guides on the surface.

### IK Guides

By default, when manipulating guides using the mouse, they use an IK (Inverse Kinematic) method of handling the guide chains which keeps all knots at an equal distance apart. If you wish to override this method, uncheck the 'IK' option. You will now be able to move knots anywhere you desire.



## FiberFX Strand Modeler > Guides Tab

### Interpolation

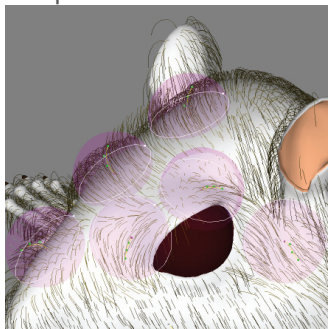
Guides have several 'Interpolation' methods available to them to further refine their area of influence on the fibers. Clicking the root knot on a guide changes its interpolation method (the color also changes to show which mode you're in) and if applicable the radius of influence the interpolation has.



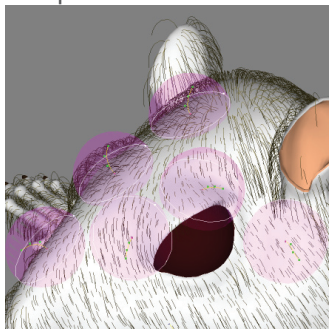
Sharp Wide



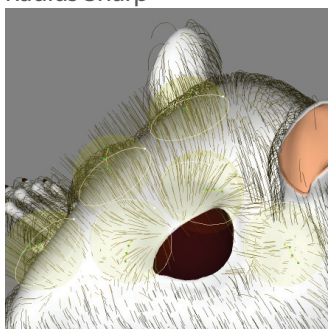
Interpolate Wide



Radius Sharp



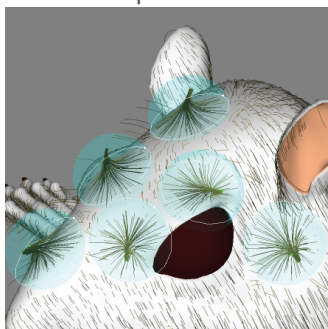
Radius Sharp w/fx



Radius Interpolated



Radius Interpolated w/fx



Bundled

### Radius

Sets the radius of influence for interpolation modes that support it.

### Bundle %

The bundle percentage pulls all fibers in the radius of influence together forming a ponytail.

### Bundle Bias

Sets how far along the fibers length they start to bundle together. At 0% the bundling starts nearer the tip of the fibers, at 100% bundling starts nearer the root.





## FiberFX Strand Modeler > Random Tab

This tab contains controls for randomizing the appearance of the fibers to create a shaggier look.

### Length

Randomizes the lengths of the fibers.

### Jitter X

Randomizes the X axis direction the fiber as it leave the objects surface.

### Jitter Y

Randomizes the Y axis direction the fiber as it leave the objects surface.

### Jitter Z

Randomizes the Z axis direction the fiber as it leave the objects surface.

## FiberFX Strand Modeler > Gravity Tab

Contains controls for simulating gravity affecting the fibers to create a more natural appearance.

### Strength

The percentage of gravity affecting the fibers. Gravity bends the fibers in the direction of the arrow in the gravity direction control box.

### Use Normals

Percentage of force pulling fibers into the direction of the surface normal. Useful for creation of surface hugging fibers instead of being pulled only in the direction of gravity.

### Gravity Direction

The gadget allows you to change the direction you want the gravity force to be directed. To modify the click and drag the arrow using the mouse.

### Reset

Reset the gravity direction arrow to point down.

### Slope

Changes the force of the gravity effect according to the slope of the surface. Fibers on upward and horizontal facing surfaces receive more of the gravity strength, whereas fibers on downward and vertical surfaces receive less of the gravity force.



## FiberFX Strand Modeler > Tools1 Tab

Tools for fiber styling can be found on this tab.

### Curl %

Percentage of curl. Use in combination with curl turns to create various curl types. These require many edges for the smoothest effect.

### Curl Turns

The quantity of curl revolutions. Set to 1 a curl would make a single 360 degree curl.

### Randomize Curling

Randomize the phase of the curl start.

### Kink

Sets the percentage of back and forth kinking like pleats or accordion fold.

### Slope Shorten

Shorten the fibers according to the slope of the surface. At 100% the fibers facing down are scaled to 0 and fibers in between are scaled accordingly.

### UV Bias

Set the directional bias to be applied over each polygon laying down according to the direction chosen. Think of this like a joystick at the North Pole. Setting this to have a specified direction lays all the fibers over into this direction. Handy for creation of parts and the whorl at the top of the head.

Bias works by laying a fiber up or down against it's underlying polygon. It can be used to create sleek lay-down types of hair. To illustrate, say you are doing a Wolfman. In the normal UV position, the hair part would be at the top of the head when contoured. All the hair would be flowing down over the face. If you changed the UV contouring setting appropriately, the same operation would produce the part at the nose. Then the hair will flow from the nose back across the face, more like animal fur.

### Reset

Reset the UV bias and directional bias back to default.

### Dir Bias

Use this to tip the "UV bias joystick" at the North Pole into a different orientation.

## FiberFX Strand Modeler > Tools2 Tab

Tools for fiber styling can be found on this tab.

### Clip Object

Select another modeler layer as a clipper object to cut the hair that intersects it. The clipper object is shown in wireframe. For instance you can use a box in another layer to clip hair into a flat top style. Imagine hair growing into a form and stopping where it intersects.

### Clip Transformation

The clip object can be moved by selecting the transform and using the 'Ctrl' key to manipulate it.

### Reset

Resets the clip object back to origin.

### Hide

Hides the clip object.

### Collision

Set the fiber collision with the surface. These are subject to effects and be aware that guides plus effects can drive fibers into the surface since by nature those are predetermined and subject to interpolation.

Guide knots themselves are subject to collisions and can be place down on a surface.

### Axis Mirror

Only available when using guides and is handy for creating a symmetrical hair part. Place the guides on one side and fibers on the other side with use them in a mirrored fashion.

### Length Vmap

Choose an existing weight map to alter the length according to the weight map values from 0 to 1.

### Density Vmap

Choose an existing weight map to alter the local fiber density according to the weight map values from 0 to 1.

### Make UVs

UV coordinates are created for the fibers, 0 at the base and 1 at the tip. Useful for weighting animated fibers using clothFX.



## FiberFX Strand Modeler > Options Tab

Contains preferences for the FiberFX Strand Modeler.

### Fiber Color

Sets the color of the fibers within the Interface. This does not affect the fibers color when rendered.

### Bkg Color (Background)

Sets the color of the viewport background within the Interface.

### Grid

Draws a X-Z grid in the viewport.

### Reset Current Surface

Reset current surface to default settings, any styling will be lost. Guides will not be deleted.

### Surface Offset

Push points off the polygon by set amount. Useful to offset single point polygons from the surface.

### Knots Only

Show only the guide knots, turning off the spline drawing.

### Ends Only

Show only the root and tip knots, hiding any knots inbetween.

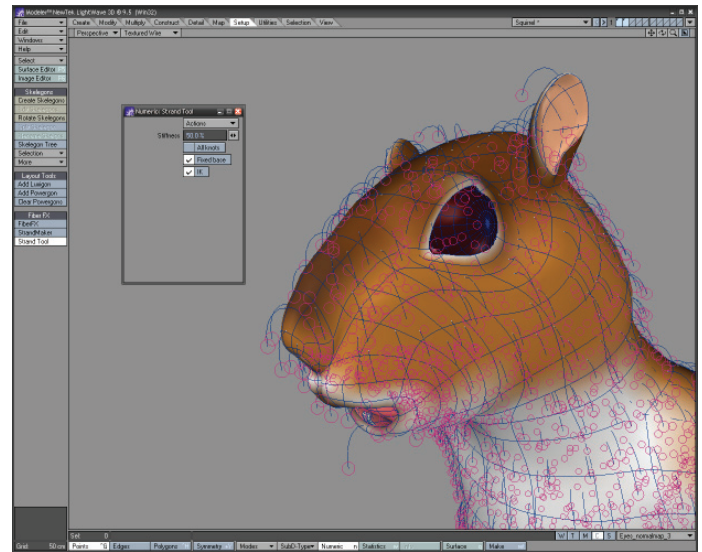
### Fat Lines

Thickens up the fibers and guides within the interface to make them more visible.

## FiberFX Strand Modeler > Strand Tool

This is a Modeler tool for adjusting fiber strands using IK after creating them in Strand Modeler. When launched the Strand Tool scans the selected layer and builds a list of fiber strands. A handle is drawn at the end of each strand and inverse kinematics is used to calculate the new position when moved.

Pressing the 'n' key brings up the numeric panel allowing you to set the IK strength. The numeric panel also lets you toggle all knots to adjust interior knots in a strand.





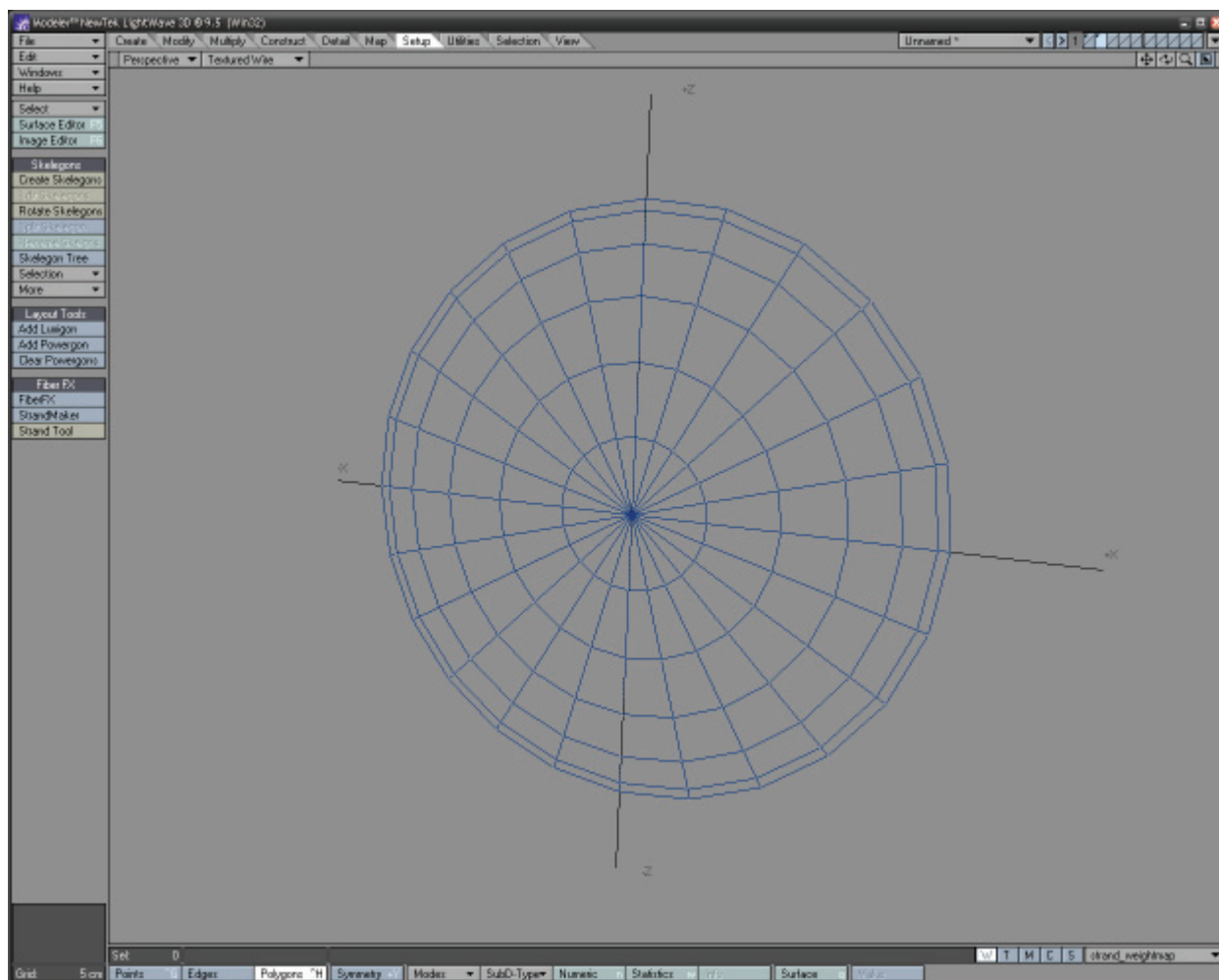
## FiberFX Strand Modeler > Strand Maker

This Modeler tool allows you to create fiber strands using standard LightWave Modeler tools. When Strand Maker is run, it scans the polygons in the current layer then moves to a new layer and builds fiber strands based off the geometry.

You can also use LightWave Modelers curve drawing tools like 'Sketch' and 'Spline Draw' to create fiber strands by hand, or convert whole objects into fibers. Any objects with edge loops will create fibers that are continuous, whereas open ended curves will create strands exactly matching them.

Once these strands have been creating, you can either use them back inside the Strand Modeler as fiber guides for further controlling your fibers, or take them into LightWave Layout for use as guides for the FiberFX Pixel Filter plugin.

In the quick example below, creating a ball object, delete one half to create a hemisphere. Flattening it in the Y-Axis and finally converting the geometry to fibers using Strand Maker could be the beginnings of a spiders web.



Converting Existing Geometry to Fiber Strands Using the Strand Maker



---

## Glossary

---



## Glossary of 3D Terms

**1-Sided** — When a polygon is created, unless otherwise set up, it will have only one side. If you were to look at a playing card, it has a front and a back. A 1-sided polygon only has a front, and therefore only one surface normal.

**2D Map** — Two-dimensional map consisting of either a bitmap or a procedural map. An object using a 2D map needs texture coordinates. See **UV** for further details.

**2-Sided** — Like a playing card. A polygon that has a front, and a back, is 2-sided. A 2-sided polygon will have two surface normals, facing opposite directions.

**3D** — A three-dimensional medium, display, or performance, especially a cinematic or graphic medium in three dimensions.

**3D Accelerator Card** — A graphics card specifically designed for 3D graphics. LightWave uses a system of 3D graphics called OpenGL, and your graphics card must support this.

**3D Map** — Three-dimensional map either built up from multiple layers of bitmaps or, more often, generated in three dimensions with a procedural texture. These are algorithms that can generate 3D maps resembling marble or wood, and when applied to an object, the grains of the marble, and the fibres of the wood, will be correctly mapped to the surface in all three dimensions. If you split a 3D-mapped cube in two halves, the cross section surface will match the neighbouring faces. A 3D map does not require texture coordinates.

**3D Object** — Anything with a position and representation in 3D space. Some objects have a special role, for instance a camera or a light, while others serve as controls for other objects, for instance splines or manipulators. The most common 3D objects are geometric objects, classified according to whether they are polygon meshes, surfaces, curves, implicit objects, or nulls.

**3D Shutter Glasses** — 3D glasses made with electronic liquid crystal shutters. They are powered by the computer they are attached to and use this power to turn on and off the liquid crystal in each of the lenses creating a 3D effect, instead of the usual 2D display a computer monitor can offer.

**3DS** — Aged file format used by Autodesk 3D Studio and discreet 3d Studio max for three-dimensional scenes. It contains geometry, textures, lights and cameras as well as animation data, but polygons cannot contain more than three points.

**Absolute Coordinates** — The location of a point in terms of distances and/or angles from a fixed origin. See **Relative coordinates**.

**Adaptive Supersampling** — Way of antialiasing an object's surface by decreasing the oversampling rate for those pixels that do not require the oversampling. The results of adaptive supersampling are slightly more localised, and computing time is often shorter than other sampling methods.

**Additive Mixing of Colored Light** — There are two sorts of mixing of colors. One is called additive, or sometimes transmissive, and refers to the fact that the more red, green and blue you

add together the nearer to white your final color will be. This is the normal light scheme for LightWave or other graphics packages with output mainly through the medium of a screen. Subtractive mixing indicates that the fewer colors you mix the nearer to white you are and is used for reflective color, such as printed material.

**Additive Opacity** — Type of opacity that adds the background color to the material color of the transparent object.

**Aggregate Object** — An object that is made up of a number of other objects. A normal aggregate object will be made up of primitives. A more complex aggregate object may be made up of primitives, other aggregate objects, or both.

**Algorithm** — A problem-solving method that involves using a multi-step process.

**Aliasing** — When referring to pictures, aliasing is the effect that occurs when a line looks jagged instead of smooth because of a contrast in colors. Usually, you can tell when this happens because the line between the colors looks very jagged, as if it were a flight of stairs, in fact it is often referred to as a "stairstepping" effect. For contrast, see **Antialiasing**.

**Alpha Channel** — One of the four channels (or components) of information that make up every pixel in an image. There are three channels for red, green, and blue (RGB) and one alpha channel. The alpha channel is really a mask — it specifies the transparency of each pixel, which allows portions of the foreground image to reveal the background when two images are overlaid.

**Alpha/Matte Image** — Generally refers to an image where the brightness of each pixel is used to cut or partially dissolve out another image. These are generally greyscale or black-and-white images, but the brightness values can also be extracted from a color image.

**Ambient Component** — Part of the reflection-illumination model. A common surface shader parameter that adds consistency to the color of an object's surface to simulate an ambient light that reaches all points in a scene. An ambient value is determined for individual surfaces. Scene ambience is multiplied with an object's ambient color. If the scene ambience is set to black, nothing alters the ambient color of an object except, of course, a light. The careful balance of ambient and direct light sources is the key to convincing lighting. Global illumination is an alternative to ambient light that is more accurate but takes longer to render.

**Ambient Map** — Allows manipulation of the ambient component of an object's reflection-illumination model. Usually the ambient component is given a value near that of the diffuse component.

**Ambient Light** — All-directional light illuminating every object uniformly from all sides.

**Anaglyph** — Moving or still pictures in contrasting colors that appear three-dimensional when superimposed. Anaglyph works well for printed matter or computer display, but color problems inherent in television displays (both PAL and NTSC) result in poor 3D broadcasts.

**Anamorphic Distort** — An option referring to the width of a lens flare. When selected, the larger the distort factor, the wider the lens flare will become.





**Angle of Incidence** — The relative angle between a lit surface and the light source. The more the surface is turned away from the light source, the less light it receives and the darker it becomes. When the angle of incidence is 90 degrees, the light shines directly on the surface and it is illuminated with maximum intensity.

**Animate, Animation** — The movement of elements through time and space. Also, the process of creating and recording images that change over time. Everything in a scene is represented by numeric values and, as such, animation is also the process of changing these values — position, color, or any other property — over time. A method of creating the illusion of life or movement in inanimate objects or drawings. Through animation the artist's drawing comes to life. The most well known works are cartoon comedies, like *Ren & Stimpy* or *The Simpsons*.

**Animatics** — Preliminary animated versions of a final video or film presentation.

**Animation Channel** — Animation Channel refers to the different position, rotation, and scaling settings an item can have in Layout. It can also refer to other envelope elements like light intensity. See also **motion channel**.

**Annotation** — The process of inserting text or a special note, explanation or to provide relevant detail to a surface, a rig or a point in your scene in LightWave.

**Antialiasing** — A method for blending harsh contours and preventing staircasing or stairstepping. It is achieved by taking the surrounding areas into account when assigning a color value to pixels lying on an object's contour.

**Antisymmetry Surface Restraint** — The restraint of a surface tangent to the surface. This implies that the structure is symmetrical about this plane, and the load on the implied symmetrical part is equal to, but in a direction opposite to, the modelled part.

**Aperture** — The opening size of a camera lens. The greater the aperture, the smaller the depth of field and the greater the amount of light entering the lens.

**API** — Abbreviation for application programming interface.

**Arc** — Curved open element with a constant radius around a single center point. Section of a circle.

**Area Light** — A special kind of point or spotlight. The rays emanate from a geometric area instead of a single point (entire surface uniformly emits light). This is useful for creating soft shadows with both an umbra (the full shadow) and a penumbra (the partial shadow).

**Array** — A set of elements put together into a single entity. A pixel array is an ordered set of colored elements used for display purposes. In a 3D program, the array tool is usually used to create ordered copies of an object within 3D space. This tool is so named because it creates arrays of objects (creates an ordered set consisting of multiple copies of the same object).

**Aspect Ratio** — Description of the proportion of an image by comparing its width to its height. 35 mm slides have the aspect ratio of 4:3 (1.33:1). Images become distorted if forced

into a different aspect ratio during enlargement, reduction, or transfers. It should not be confused with the pixel aspect ratio, explained further on.

**Atmosphere** — Volumetric effect that simulates reduced visibility in air over distances.

**Attenuation** — The strength of light diminishes with distance when it travels through air. The further light travels, the dimmer it becomes. In real life, light attenuates by the inverse square of the distance. If attenuation is turned on for a light, only the geometry in its proximity will be lit. Not only is this more realistic for your renderings, it also helps speed up rendering time since only the geometry close enough to be affected by the light needs calculation time. See also **Decay**.

**AVI** — Audio Video Interleave. A popular animation file format that combines video and audio.

**Axis of Motion** — In 3D space, the line that an object follows during movement.

**Axis of Rotation** — In 3D space, the line that an object rotates around.

**Axis, Axes** — Axis refers to the XYZ coordinates used as the basis for positioning elements in LightWave's 3D space. It is somewhat like the concept of left/right, up/down, and near/far.

**B Rep** — See **Boundary Representation**

**Backface Culling/Elimination** — A process included in most 3D graphics pipelines, backface culling eliminates triangles facing away from the camera. This is most efficiently performed by checking the orientation of the triangle normal in relation to the camera. The technique ignores geometry seen from behind so that only the fronts of objects that are facing the camera are rendered. Both faces of an object are rendered by default; that is, the ones whose normals are facing the camera as well as those that are not. You can choose which faces of the object you want to render as part of the rendering options: front, back, or both faces. Backface culling (rendering only the front) can improve performance because less geometry needs to be rendered.

**Background Color** — The color that occupies all space not occupied by text, images, or any other objects. LightWave's default background color is black.

**Ball** — (Sphere) A 3D circle or oval created by user-defined dimensions and settings.

**Bandwidth** — How much information a network can carry. Think of the network as a highway, and each message as a car. The more lanes in the highway, and the higher the speed limit, the more traffic it can carry. So the wider the bandwidth of a network, and the faster its speed, the more information it can carry.

**Banking** — This is when an object following a path rotates about the path when it changes direction.

**Barn Doors** — The flaps on the front of a movie light to assist in limiting the beam of light.



**Baud** — Bits per second. Hence kilobaud or Kbaud, thousands of bits per second. The technical meaning is 'level transitions per second'; this coincides with bps only for two-level modulation with no framing or stop bits.

**Bend** — To deviate from a straight line or position: The lane bends to the right at the bridge. To assume a curved, crooked, or angular form or direction: The saplings bent in the wind.

**Bevel** — A method of eliminating sharp edges from objects by extending an object's faces to round off corners.

**Bezier Curve** — A technique for creating curves that was attributed to and named after a French engineer, Pierre Bézier, who used them for the body design of Renault's cars in the 1970s.

**Bilinear Filtering** — Blurring the pixels in a bitmap when it is zoomed in so that it seems smoother than it really is.

**Bilinear Intensity Calculation** — A high-speed algorithm for generating shaded faces. Used in Gouraud shading and Phong shading.

**Binary Space Partition (BSP)** — Also known as BSP, this is a technique used in real-time rendering for resolving in which order polygons should be drawn. The technique requires that a so-called BSP tree is built before the scene may be rendered. As this build process is very costly in terms of execution speed, BSP trees cannot usually be calculated in real-time and thus essentially only support highly complex yet static 3D worlds.

**Bit** — The building blocks of computer data. Has either the value of 1 or 0 (current or no current). Bits can be grouped together to carry larger values.

**Bitmap** — Two-dimensional monochrome raster image. A bitmap is a black and white image marking boundaries. It is often used for clip maps in LightWave.

**Blade** — A thin plane placed in front of a light to cast a shadow, taking light off of an object. A device to create a shadow.

**Blending** — Mixing of two (or more) textures into one final texture that is displayed in rendering.

**Blitting** — The copying of a virtual frame buffer to the displaying screen.

**Bone Hierarchy** — Bones can be arranged to build a Bone Hierarchy, also called a Skeleton. The hierarchy defines how the movement of one bone affects other bones (up and down the hierarchy). If you then also add Constraints to the bone hierarchy, you have a Rig.

**Bones** — For any object, you can define a skeletal system composed of bones. By moving bones around, you can change the shape of an object.

**Boolean** — A mathematical system developed by English mathematician George Boole that expresses logical relationships between things. The results in a Boolean operation can be either true or false. Boolean is used in 3D to add, subtract, and other operations that involve Boolean calculations.

**Boolean Operations** — **Modeling** technique that uses two overlapping objects to create a new object. There are three kinds of boolean operations: subtraction, union and intersection. By taking the first shape and subtracting/unifying/intersecting it to the second — a new shape is created.

**Boom Shot** — A camera move. Usually describes a shot in which the camera is mounted on a crane. The camera can move in all axes of movement.

**Boundary Representation (B Rep)** — A polygonal mesh representation. A polygonal mesh is, most commonly, a simplification of a shape using facets to describe curvatures. Its surface, or boundary, is built up from several faces that describe the shape. If it is a polyhedron the polygon model can be identical to the shape, whereas an organic shape is represented by a more or less simplified version that mimics the curvature using facets with variable density.

**Bounding Box** — A cubic shape that exactly circumscribes a (more complex) 3D model and is used to optimise three-space calculations like ray tracing. By representing a more complex shape with a box, possible ray intersections can be investigated much more swiftly. Also used to represent complex objects for proxy animation and setup to speed up operations.

**Bounding Volume** — A way of speeding up ray tracing operations involving intersection calculations, by inscribing a complex mesh in a considerably less complex shape like a box or sphere. Often used when rendering must be done in a short amount of time. Instead of having to check the intersection of a more complex mesh, like a space ship or a teapot, the bounding box works as a stand-in, with the same maximum height, width and length as the mesh it substitutes for. Therefore a possible ray intersection can be either ruled out (if the ray doesn't pass through the bounding box, it doesn't pass through the mesh either), or let a more time-consuming algorithm take over working with the complex mesh instead.

**Box** — (Cube) A six-sided 3D object that can be thought of as a 3D square or rectangle. Boxes are created based on user-defined input as to the dimensions and locations desired.

**BSP** — See **Binary Space Partition**.

**B-Spline** — A free-form curve that is defined with parameters in which each separate vertex on the curve has an influence over a portion of the curve. In 3D, B-splines allow a user to control a curved line on two axes at once.

**Bump Array** — The purpose of a bump array is to create an ordered series of bumps in a surface. This tool means exactly what its name implies - an array of bumps. See also **Array**.



- Bump Map** — Creates the illusion of three-dimensionality of a surface (protrusions and cavities) by recalculating the normals of the object, without changing the mesh itself. It is very common in 3D renderings and suitable for creating effects like wrinkles, creases, crumples, cracks, seams etc. The silhouette of a bump mapped object is a give-away since, in these areas, it is obvious that the mesh is left unaffected (if trying to create an orange by using a perfect sphere with an orange peel texture applied to it for bump mapping it will still have an impeccably round silhouette). In LightWave areas in a bump map that are black are unaffected and areas that are white are raised.
- Byte** — 8 bits. Multiples of bytes make up the terms kilobyte (1024 bytes), megabytes (1024 kilobytes) and gigabyte (1024 megabytes).
- CAD** — Computer Aided Drafting (or Design); A system that lets a designer use a computer screen instead of a drafting table to make plans and blueprints. However, CAD packages don't often have superb rendering abilities, so packages like LightWave are turned to, for their abilities.
- CAM** — Computer Aided Manufacturing; the process of using a computer to create a physical product from a computer-created design. CAM is usually used to control robots that perform tasks that would be tedious or dangerous to human workers. See also **Rapid Prototyping**.
- Camera** — An apparatus for taking photographs, generally consisting of a lightproof enclosure having an aperture with a shuttered lens through which the image of an object is focused and recorded on a photosensitive film or plate. Digital cameras use CCDs to focus light and create a digital picture that can be seen and transferred to a computer immediately. In LightWave, the camera is the conduit through which objects and scenes are turned into still images or animations.
- Capping** — Surface closing the upper and/or bottom side of an object such as a cylinder.
- Cartesian Coordinate** — Mathematical representation of Euclidean space. Every point can be described by three coordinates (X, Y, Z) representing the position along the orthogonal X, Y, and Z axes. The point (0, 0, 0) is called the origin, which is the global center of the 3D world.
- Cartesian Space** — A space in which positions are denoted by a three-coordinate system (x, y, and z coordinates) relating to a central origin (0,0,0).
- Catacorner** — Slanted across a polygon on a diagonal line; "set off in a catty-corner direction across the vacant lot". syn: cata-cornered, catercorner, cater-cornered, catty-corner, catty-cornered, kitty-corner, kitty-cornered.
- Cattiwompus** — Weird. Mixed up. Unusual. Distorted.
- Caustics** — Light pattern created by specular reflection or refraction of light, such as the patterns of light on the bottom of a swimming pool, or light through a glass of wine.
- CD** — Compact Disc storage media. Also the chemical symbol for Cadmium and the sticker on the back of a car with the diplomatic corps.
- Center of Projection** — The point in perspective projection where all projectors intersect.
- Center of the World** — Is the absolute center of a 3D space, represented by X, Y, and Z points (0, 0, 0). Also referred to as the Origin.
- CenterPoint** — A point that represents the center of an object. This point is used in some programs for a point of reference for rotation and position. The center point of a polygon is where the line representing the normal comes out from.
- Chamfer** — To cut off the edges of the geometry with a planar cut, creating a more blunt shape, typically at a 45 degree angle. A bevelled edge or corner between two intersecting lines or surfaces.
- Child** — An object whose movements are influenced by another object, called the "parent".
- Chord Length Parametrisation** — See **Non-Uniform Parametrisation**.
- Clean Modeling** — Refers to the practice of removing geometry from a model that is not wanted or needed. Also refers to the use of proper geometry construction techniques, such as creating continuous surfaces, minimising narrow faces, and avoiding small corner angles, that facilitates downstream processes.
- Clipping** — More often than not, much of the graphics drawn for a specific scene does not fit into the viewport of the camera. Accordingly, those which fall outside of the viewport must be clipped so as they are not drawn. Depending on the nature of the application, there are two kinds of clipping: 2D and 3D. The earlier simply compares each pixel against the extents of the rendering viewport, while the latter technique uses the six sides of the view frustum to determine whether a 3D vertex is inside the viewport or not.
- Clone** — This tool creates copies of an object based on user-defined parameters for offset, motions, morphing, shadows, etc. This tool can be used to make ordered sets of objects, but is different from the array command because not all new objects need be exactly the same as the original.
- Cloud of Points, or Point Cloud** — A set of x-y-z coordinates obtained from a 3D scanner or digitiser. The data can then be turned into a continuous surface and used as a 3D model.
- CODEC** — Short for "COmpressor/DECompressor". This is the term used to reference the way that software programs handle different movie files, such as Quick Time, AVI, etc. The CODEC can control image quality, and can assign the amount of space given to the movie file. First, a multimedia publisher uses a codec to squeeze more sound and video into less file space. These compressed files are easier to fit on a CD-ROM and transfer to your computer. Then, your computer uses a codec to expand these files back to their original size and replay them on screen.
- Coincidence** — Refers to geometry that occupies the same spatial location. For example, coincident vertices are points that occupy the same x, y, and z coordinates. Coincident lines can have differing lengths while one occupies the same location as the other.



**Color Bleeding** — When the color from one surface reflects onto another surface.

**Color Depth** — The number of bits used to represent a color. For example an 8-bit image uses  $2^8=256$  colors. The bits build up the three primary colors red, green and blue.

The bits are divided into red, green and blue (optionally an alpha channel as well).  
For example a 16-bit color could look like this R: 4-bit (16), G: 4-bit (16), B: 4-bit (16), Alpha: 4-bit (16) — together they add up to 16-bits. The number of bits can also be unevenly divided (R:5, G:5, B:5 Alpha:1).

This is why a GIF (max 8-bit=256 colors) only sports 128 colors if it is transparent (1 bit is used to represent transparency in the alpha channel, 7-bits =  $7^2=128$ ).

The following table indicates the number of colors an image can have.

8-bit =  $2^8 = 256$   
16-bit =  $2^{16} = 65536$   
24-bit =  $2^{24} = 16$  million  
32-bit =  $2^{32} = 4.3$  billion

You should also be aware of FP, or HDR images.

**Color Model** — A system used to specify colors. In LightWave, you can set color according to the following color models: RGB (red, green, blue), HLS (hue, lightness, saturation), HSV (hue, saturation, value) or integer values.

**Color Keying** — An old technique of combining two images by replacing one color of the destination image by the corresponding pixels of the source image.

**Column Interleaved Format** — The 3D image format used by the VRex VR-4200 projector ([www.vrex.com](http://www.vrex.com)). Left and right view image data are encoded on alternate columns of the display.

**Compiled Vertex Array (CVA)** — Array of geometry data on a vertex level that is optimized (compiled) for faster access by the graphics card. (Note that this is an OpenGL term, and is known by other names in other APIs.

**Compositing** — A layering technique that places one image on top of another, properly taking into account transparent pixels, apparent depth, shadowing and other elements that make up an image.

**Concentric** — Having a common center or origin point with varying radii.

**Cone Angle** — The angle at the peak of a cone.

**Conic** — Element having the form of a cone.

**Conic Section** — Curve formed by the intersection of a plane with a cone.

**Constraints** — Values in a geometric model that define relationships, i.e. a line is tangent to a circle. Constraints are often used to drive parametric or variational geometry-based systems; the algorithms used to work with constraints are known as constraint management.

**Continuous LoD** — Short for continuous Level-of-Detail, this method is based on the observation that 3D objects located far off in the distance may be approximated by simpler versions without loss of visual quality, thus increasing the rendering performance. The “continuous” refers to having the algorithm constantly recompute the detail level of the 3D object depending on the distance to the camera instead of having a pre-computed set of objects to choose from. Computationally expensive, this method is most often used in height field rendering applications. LightWave can approximate continuous LoD by using an Object list.

**Control Mesh** — A cage of points used to shape SubPatches.

**Convex Volume** — A convex volume can be defined as a volume whose every corner can be visible from all other corners in the same volume. Another way of defining the convexity is that all faces in the volume will be lit by a point light located anywhere within the volume.

**Cookaloris, Cookie** — A device put in front of a light, to break the light up. Common cookaloris look like leaves on trees, or blinds on windows.

**Coons Patch** — A free-form surface that is determined by the four curves that define its edges.

**Coplanar** — Refers to two or more entities that lie on the same plane. Two planar surfaces, for example, that lie on the same 3-dimensional plane are considered coplanar. If these coplanar surfaces share a common edge, it is recommended that they be joined into a single surface.

**Cross Product** — Using two vectors to calculate their normal.

**Cross-Section** — A view of the interior of an object as it is sliced along a plane.

**Cubic Image Map** — One of the many methods of applying a texture to a surface. This method applies the texture to an object as it would apply the texture to a cube. There are many other methods of texturing objects, such as Planar and Cylindrical image mapping.

**Curvature Continuity** — A curvature continuity with smooth transition of the edges of two meeting surfaces (the highlights of the two surfaces blend together seamlessly, forming the illusion of a single shape). If a curve (or surface) has tangent continuity and both the 2D curves (or 3D surfaces) have the same radius a very smooth transition is created with curvature continuity. Curvature is defined as  $1/\text{radius}$ . Hence, small radius equals high curvature.

**Curve** — In computer graphics, there are different ways of representing a curve, such as NURBS and Bezier curves. See also **NURBS** and **Bezier**.

**Curve Parametrisation** — See Parametrisation.

**CVA** — See **Compiled Vertex Array**.

**Cylindrical Image Map** — One of the many methods of applying a texture to a surface. This method applies the texture to an object as it would apply the texture to a cylinder. There are many other methods of texturing objects, such as Cubic and Planar image mapping.





**Decay** — Phenomenon where light intensity decreases with distance. The further away from the light source, the less intense are its rays. In the real world the decay is proportional to the inversed square of the distance (quadratic decay), but there is also directional (one-dimensional) decay (slower than in real life) as well as cubic decay (faster than in real life). See also **Attenuation**.

**Decompression** — Process of returning a compressed file to its full size.

**Default Unit** — The Default unit is the unit of measure (ex. meter, feet, etc.) that is assumed, usually when no unit of measure is entered with the numeric data. In Layout, it is determined by the setting on the General Options Tab of the Preferences panel. In Modeler, the setting is on the Display Options panel.

**Depth Buffer** — See **Z-Buffer**.

**Depth Cueing** — The process of reducing the apparent sharpness of an object the farther away it is from the viewer or camera. This often enhances the perception of depth.

**Depth of Field (DoF)** — The total distance, on either side of the point of focus, which, when viewed from an appropriate distance, appears sharp in the final print.

**Depth Sorting** — Sorting all triangles in the world depending on diminishing depth (lower and lower z-value) so when rendered, the triangle closest to the viewer obscures those behind it.

**Deskew** — Process used to remove skew or distortion through a small angle rotation.

**Diffuse Component** — Part of the reflection-illumination model. The diffuse is concerned with the amount of light that is reflected back.

**Diffuse Light** — A component of the reflective model that is the result of direct illumination.

**Diffuse Map** — Replaces the diffuse component of the reflection-illumination model, basically giving the illusion of being painted onto the surface. To create a material resembling wood or marble, this map is used. Generally, when you talk about the “texture map” in an application, this is the map actually referred to.

**Dimension** — A measure of spatial extent, especially width, height, or length.

**Directional Light** — See **Distant Light**.

**Director of Photography (DP)** — Person on set that determines how to photograph the movie.

**Disc** — Shape that is referred to in mathematics as a cylinder. This shape is composed of two circular or oval-shaped bases and the space contained between those bases. In other words, a disc is like a stack of circles with set parameters defined by you.

**Displacement Map** — Can be used to modify the actual mesh (as opposed to the bump map) to create wrinkles, creases, crumples etc. The displacement map will need a more complex mesh to create the same effect as bump mapping, but has the advantage of allowing more thorough close-ups, since the surface is actually deformed and not just simulated as being so.

**Display Types** — Ways of displaying objects in a viewport. Display types are available only for geometry views. The available display types are Bounding Box, Vertices, Wireframe, Front Face Wireframe, Shaded Solid, Textured Shaded Solid and Textured Shaded Solid Wireframe. Display types do not determine the quality of the final render.

**Distant Light** — A light with color, intensity and direction. All rays emitted from a distant light are parallel, and therefore it has no obvious source. Distant lights can be used to simulate point lights from a great distance (whose rays can be approximated to be parallel), for example the sun. The intensity from a distant light does not decay.

**Dithering** — Creating the impression of having more color on the screen than there actually are by plotting pixels (with a limited amount) of different colors next to each other.

**DoF** — See **Depth of Field**.

**Dolly** — To move the camera along its line of sight (in a straight line following the imaginary path between the actual camera and its target point).

**Dongle** — A hardware lock used to prevent the piracy of LightWave.

**Dopesheet, Dopetrack** — Animation tools in LightWave that allow you to better organise keyframes.

**DPI** — Dots per Inch. In a bitmapped image, the number of dots that exist within each inch of the image. This number remains constant, so when you make an image larger, the quality decreases, but when you make the image smaller, it appears to increase.

**Double Buffering** — This is the process of using two frame buffers for smooth animation. While the image of the first buffer is being displayed, the graphics controller can use the second buffer to build or render the next image. Once the second image is completed, the buffers are switched. Thus, the result is the appearance of smooth animation because only complete images are displayed, and the process of drawing is not shown. You can often now see triple buffering in graphics cards to allow an extra buffer for the next image in case there is a problem.

**DP** — See **Director of Photography**.

**DVD** — A high-density compact disc for storing large amounts of data, especially high-resolution audio-visual material. DVDs used solely for computers are commonly referred to as DVD-ROM.



**DWG** — AutoCAD native file format. It can contain 3D data, but is hard to convert to a LightWave-native format because of its construction. A DWG file is parametric, that is to say it does not contain the objects themselves, but rather instructions on how to build the objects. This makes it hard to translate if you do not possess a licence of AutoCAD. The solution is to either get one or get your client to supply you the object in a different format, preferably OBJ.

**Easing** — Reduction in the acceleration or deceleration of motion to present a smoother, more continuous movement. The shape of a function curve can reflect this when using spline interpolation.

**Edge** — Straight line connecting two points on a polygon.

**Edge Loop** — Particular method of modeling organic shapes with the edges of polygons creating a loop or a flow around circular features, like the eyes and the mouth for example.

**Endomorph** — Object containing one or more Morph Maps.

**Envelope** — Way of animating a particular value over time using a graphical input mode.

**Environment Map** — Map often used to simulate (faking) reflection of the surrounding world without using ray tracing.

**Euler Angles** — Euler angles are one of the simplest methods of representing 3D rotations, and generally also the easiest to visualize. An object's rotation can be specified in terms of its yaw, pitch and roll, or rotation around the Y, X and Z axis, respectively. Euler angles suffer from singularities in the form of so-called Gimbal lock, however, and are also difficult to smoothly interpolate for keyframe animation.

**Expression** — Mathematical expressions that allow you to change the animation of an object. You can also create constraints between objects using expressions or create conditional animation. Expressions are very powerful for creating precise animation and to create automated animation of things such as wheels.

**External Attributes** — The position of the camera and the direction it is pointing in.

**Extrude** — Creating a three-dimensional object from a two-dimensional shape by adding a third dimension to it. You can also do this along a motion path or spline.

**Face** — The shape made up by the bounding point making a polygon. Faces can have as many vertices as wanted, but only polygons having a shape of three or four vertices can be made into subdivision surfaces.

**Face Normal** — Also just known as the normal, this is a line perpendicular to the face that also describes which way the face is pointing in a one-sided polygon.

**Falloff** — The volume starting at the outer rim of a spotlight's hotspot, decaying from full intensity at the start to zero intensity at the outermost rim of the spotlight. The less the difference (in angles) between the hotspot/falloff, the crisper the shadows. If the falloff angle is much larger than the hotspot angle, the boundaries of the area lit up by the spotlight will be fuzzy.

**Field of View (FOV)** — The angle of the view frustum. The wider the FOV, the more you see of the scene. Human eyes have a FOV of about 50 degrees, and normally virtual reality application use similar values to resemble real life.

**Field Rendering** — An option that causes the program to render two interlaced fields of information. This is in contrast to rendering only one (full frame) and makes moving objects appear to move more smoothly. Used for projects that will play back on television monitors that display 50 or 60 interlaced frames per second. Fielded animation is not useful for animations designed to be displayed on computer monitors. See **Fields**.

**Field Sequential 3D Video** — The most common format for 3D video. Left and right image data are encoded on alternate fields of the standard video signal.

**Fields** — Interlaced images (video) consist of two fields combined into one frame. Each field contains half the scan lines (either even or odd) and is a separate pass. This is more common to render to for TV broadcast. Moving items horizontally will strobe without rendering to fields.

**Fill Light** — Additional light sources assisting the key light in a scene. Usually they are less intense than the key light and created using point light or spotlight.

**Fillet** — To round off the edges of an object with a round shape. Think "router", use "Rounder" in Modeler to achieve it.

**Fill-rate** — The amount of pixels from a texturemap (texels) that are rendered per time unit. Measured in texels/second.

**Filter Opacity** — Type of opacity that uses a color to simulate object opacity.

**Finger** — A small strip placed in front of a light to cast a discrete shadow.

**FK** — Forward Kinematics. Positioning a .gure by specifying joint angles, like posing a toy action .gure.

**Flag** — A large device placed in front of a light to create a shadow, creating a large shaded area.

**Flat Shading** — Shading technique where all individual faces in a mesh are assigned a single color value based on the orientation of their face normals.

**Flatness** — Flatness is used as a threshold in determining if a polygon is non-planar. A flatness of 0 percent means the polygon is absolutely flat. Flatness is computed as a percentage deviation from a triangle (the "ideal plane") formed from the first two and last vertices of a polygon. All of the other points are measured relative to this plane. The largest deviation is divided by the total size of the polygon to get a percentage that is the flatness value. For example, if a polygon is 1 meter wide, .5% flatness means that no point will be outside the ideal plane of the polygon by more than 5 millimeters. (1 x .005)

**Floating Point (FP) Images** — Refers to images that do not use standard color depth models to represent the colors contained in them, but rather an expression of the floating point value of a color changing from 0 for black up to 1 for the brightest point in the image. A mid-grey in such an image would be represented by R: 0.5, G: 0.5, B: 0.5.





- Focal Length** — The distance between the lens and the light-sensitive surface on the backplane (the film in real-world cameras). The lower the focal length, the more of the scene is visible. Focal lengths less than 50 mm is called wide angle, while lengths greater than 50 mm is referred to as telephoto lenses. The longer the lens, the narrower the field of view. Distant details become more visible. The shorter the lens, the wider the FOV. More of the environment is visible in the rendered image. To simulate the human eye, you can use values of about 50 mm.
- Fog** — Simple yet effective tool often used in real-time graphics to obscure the far plane, thus bounding the viewing distance of the application. There are essentially three types of fog: table-based, vertex-based, and volumetric. Fog values may also follow linear or exponential curves.
- Foreground Image** — The image closest to the camera.
- Foreshortening** — The apparent effect of viewing an object on its long axis that makes it seem shorter. For instance, an arm pointing directly at the camera seems to lose its length as does a road going directly away toward the horizon.
- Formula-Defined Shapes** — Refers to shapes that are defined by using one or more equations. This includes complex shapes such as aesthetic bottles, or simple shapes such hyperbolic paraboloids, oblate spheroids, prolate spheroids, or ellipsoids.
- FOV** — See **Field of View**.
- FPS** — Frames Per Second. The main unit of measure that is used to describe graphics and video performance.
- Frame** — One image out of many that define an animation. There are 24 frames per second in film, 25 frames per second in PAL video, and approximately 30 frames per second in NTSC video.
- Frame-Buffer** — The memory a computer uses to hold one or more frames for later use.
- Frame-Rate** — The speed at which a frame of animation is shown, usually expressed in frames per second. European TV is at 25 frames per second, US is typically 29.97 frames and movies are projected at 24 frames.
- Freeze** — To convert from vector or interpolated geometry (splines, NURBS, subdivision surfaces) to pure polygons. Even if the renderer supports NURBS or subdivision surfaces, this freezing happens at render time, and is usually definable to the level of polygon creation by you.
- Frustum** — The part of a solid, such as a cone or pyramid, between two parallel planes cutting the solid, especially the section between the base and a plane parallel to the base. See **View Frustum**.
- FX** — Shorthand term for effects.
- GCore (Geometry Core)** — Engine in LightWave that handles all animation and Modeling tools.
- Generic Primitive** — Simple 3D objects that most 3D programs can create easily. These objects typically consist of spheres, cylinders, cubes, and cones.
- Geometry** — Positional layout of points and polygons for an object. These points are usually seen with objects that can be rendered. For example, a cube's geometry is composed of eight points. A curve has geometry since it is composed of one or more points, whereas nulls have no geometry.
- Gimbal-Lock** — What happens when two axes of rotation line up, thereby making three-dimensional rotation impossible. As an example, take any object with neutral rotation (0 degrees on heading, pitch, and bank) and rotate the pitch 90 degrees. Now, try to rotate the bank. This is gimbal-lock.
- Gizmo** — See **Null**.
- Global Illumination** — Unlike local illumination, this method of generating images supports effects not only linked directly to the light sources themselves. In real life, the intensity of a surface depends not only on direct illumination from the light source itself, but also from indirect illumination from surfaces being lit.
- Ray tracing can cast shadows from an object onto a surface, allowing objects to be reflected in shiny surfaces or refracted in transparent materials. Radiosity is the effect of reflected light. If you have spotlights projected at the ceiling in a white room, the light will bounce back and light up the entire room. However, this can only happen if the renderer supports radiosity (as LightWave does) or other similar techniques.
- Glossiness** — Affects how spread out across a surface a lighting highlight is. Low glossiness makes a spread out highlight while high glossiness creates a more central, pinpointed highlight.
- Glossiness Map** — An image to control the glossiness of a surface. Bright values in the image indicate more glossiness, dark values less.
- Glow** — Optical light effect that looks like a fuzzy disc around the center of a light source.
- Goal** — An object used in IK to create a point where an object will always reach for. This is used to make objects appear to have realistic motion.
- Gouraud Shading** — Developed by Henri Gouraud in 1971, this is a fast incremental shading technique using bilinear intensity calculation to create smooth transitions between the vertices in a triangle. It is most often used for lighting purposes by computing the vertex normals in the polygon, calculating the light values for each vertex, and then Gouraud shading the polygon. Even though it has obvious advantages over flat shading, the facets in the mesh can still be discerned. The placement of the highlight depends on the underlying polygons.
- GUI** — Graphical User Interface. The graphical interpreter between man and computer allows a more intuitive interaction with the computer. The window maker in UNIX, and Windows for the PC are both GUIs. This way you don't have to be computer literate to the same extent as if you should have to type all commands you wanted the computer to perform.



**Greeblies** — English slang. This describes the non-such little details on objects, usually mechanical objects. Those details which can be found on spaceships, in engine rooms, etc. You can also use the words “didges”, “nurnies” and “doohickies”.

**Halo** — Optical light effect that forms concentric circles around the center of a lightsource. Often clearly visible around street lights after a rainy day.

**Hidden** — Any element that is not shown in the current rendering of the scene but that still exists.

**Hidden Surface Removal** — Algorithm for removing obscured polygons in a three-dimensional view space. As opposed to the faster algorithm backface culling, the hidden surface removal algorithm is able to sort out those polygons that are obscured by another object. Another way of finding an obscured polygon is the z-buffer.

**Hierarchy** — A way of defining objects in relationship to each other (using a parent-child or tree analogy). This relationship means that transformations, deformations, and any other property of the parent object affect all child objects. This allows separately modelled objects to be used in a scene as a single functional unit. The movement of a parent affects the movement of the child, but you can move the child without affecting the parent.

**HDRI** — High Dynamic Range Image. An image with a wide intensity range between the brightest and darkest pixels. In typical 8/24-bit images, the maximum possible intensity range is 255 times brighter than the darkest grey pixel (with a value of 1). Natural scenes and images rendered with radiosity can have dynamic ranges from 10 to 10,000 times greater than this. Recording this information requires use of an image format with higher precision - such as LightWave's native .FLX format.

**Highlight** — Reflection of a light source on an object's surface. The size of the highlight (the area that shows the light source reflection) depends on the angle. Consequently, multiple light sources result in multiple highlights. This is also the Specularity.

**HLS Color Model** — Hue, Lightness and Saturation: the three components of the HLS color model. Hue refers to the position of the color in the spectrum, such as red, yellow, or green. Lightness is the amount of white mixed in a color, such as the difference between a pure red and pink. Saturation is the purity of the color, such as the difference between a pure red and a dusty rose - low saturation means that there is grey in the color.

**Hotspot** — The inner intense cone of light emanating from a spotlight.

**HSV Color Model** — Hue, Saturation, Value: the three components of the HSV color model. This color model defines the hue and saturation similar to the HLS model. Value is similar to lightness, as in HLS; however, a value of 1 represents a pure color when saturation is 1, while a lightness of 1 yields white no matter what the saturation. In both systems, 0 is black.

**Hub, The** — Module in LightWave that allows the Layout and Modeler modules to synchronise information. It uses the TCP/IP protocol to transfer information between modules.

**Hue** — The position of the color in the spectrum that describes the tone or tint of a color, such as red, yellow, or blue.

**HyperVoxel** — Voxels are volumetric rendering effects. HyperVoxels are voxels that are applied to nulls, points, or objects.

**IK** — Inverse Kinematics. The process of determining the motion of joints in a hierarchical 3D object given the desired start and end points, all the while obeying the laws of kinematics. Think of it like the strings on a marionette puppet.

**Image instance** — A copy or instance of a source image. Each time you use a source image, an instance of it is created. You can have as many instances of the same source as you need. You can then edit, crop, or even blur the instance without affecting the original source image.

**Image Map** — An image that is applied to an object's surface.

**Incandescence** — The emission of visible light by a hot object. In LightWave, this is the luminosity channel.

**Incremental Shading** — See Interpolative Shading.

**Indirect Illumination** — Light that bounces off one surface and illuminates another surface. This can only happen if the renderer supports radiosity. The LightWave renderer supports radiosity.

**Intelligentities** — Refers to LightWave's object format. The object format can contain morphs, multiple layers, and independent pivot points on a per layer basis.

**Intensity** — The strength at which the light source illuminates objects in the scene.

**Interference Checking** — The process of identifying if and where two or more pieces of geometry (usually solids) intersect. When moving parts are involved, a kinematics analysis is used to detect interferences.

**Internal attributes** — Properties of the camera such as depth of field and line-of-sight - compare with External Attributes.

**Interpolation** — Process used to estimate an unknown value between two or more known values. In animation, interpolation is the process used to calculate values at frames between two keyframes in a sequence.

**Interreflection** — When a reflective object reflects another reflective object. For example, if you place two mirrors in front of each other, the first one will display the second one, who, in turn, shows the first one. In real-life, there is virtually no upper limit of how many interreflections may occur, whereas in 3D rendering, one must set an upper limit to be able to render the scene. The default value for LightWave is 16, but it can be lowered to 0, if desired, or raised to 24 at a cost in increased rendering time.

**IR Transmitter** — A device that sends synchronisation signals to wireless shutter glasses.

**Isometric view** — Standard view in a 3D design where the top, front, and right side faces of a cube are equally inclined to the screen surface.



- Item** — An item in Layout refers to an object, bone, light, or camera.
- JPEG** (Joint Photographic Experts Group) — A widely accepted, international standard for compression of color images.
- JPS** — Stereoscopic JPEG file. JPS refers to a stereoscopic image file format that is based on JPEG compression. Used by DepthCharge & other stereoscopic imaging applications.
- Junkyard** — A special directory used by some studios to hold mechanical and non-organic pre-modelled parts.
- Keyframe** — (Key) A frame for which you define an animation channel(s) (e.g., position or rotation) for an item in Layout. Animations are composed of a beginning keyframe, an ending keyframe and usually some number of keyframes in between. See also **Tween**.
- Key-Light** — Dominant light source in a scene, normally created with a spotlight.
- Kinematics** — The properties of each 3D object that control its transformations. These transformation properties are used to modify the selected object's scaling (size), rotation (orientation), and translation (position) in X, Y, and Z in either local and global space. Although related, kinematics are not to be confused with inverse and forward kinematics for animation.
- Kit-Bashing** — An expression taken from model making. The practice of using model kits to give detailing to a larger project. This is still in use. It refers to the taking of models that you have already made, to use in the creation of another, perhaps even basically unrelated model.
- Lasso** — One way to perform a selection of points or polygons. This method involves drawing a loop that encircles all of the objects that need to be selected.
- Latent Surfaces** — Surfaces that are no longer visible after a Boolean or intersection operation because they lie inside or outside the solid.
- Lathe** — Creating a 3D object from a 2D shape by rotating it around an axis.
- Lattice** — Either a way of deforming object using a lattice or a way of creating outlined geometry.
- Layer** — A portion of a scene. Each layer consists of an object or multiple objects that can be edited separately from the rest of the objects in a scene. A layer is basically a building block for a scene and each layer contains separate blocks for a final model.
- Left-Handed Coordinate System** — Coordinate system where the positive part of the Z-axis goes away from the observer (from the screen).
- Lens** — Part of the camera determining the optical characteristics of the image, such as wide angle, fish eye, and depth of field.
- Lens Flare** — Optical light effect made up from a number of bright discs. If the rays from a light source reflect off the surface of a compound lens in a camera, it can generate star-like patterns on the image. Lens flares tend to be a cliché of bad CG imagery, probably because of their short rendering time and flashy appearance.
- LoD** — Level of Detail. This is a term which refers to varying the amount of detail in an object depending on the distance from the object to the camera. Example: A car for a close-up would need to have every little detail modelled into it. Chrome, bumpers, body seams, door handles, etc. But that same car, as seen from a helicopter flying over a highway, might be able to be a simple cube with an image map applied to it.
- Level-of-Detail Control** — The ability to vary the amount of details displayed in a graphics image to improve performance. For instance, at a distance, models can appear as simple 3D figures, but as users zoom in, a more detailed representation is presented.
- Light** — In LightWave, a light is generally used just like a light in real life. Lights illuminate a scene and allow you to see the objects within it. Different types of lights are distinguished: ambient light, diffuse light, point light, spotlight, etc. There are also different terms used to simulate the way material properties are illuminated: ambient component, diffuse component, specular component. Incident light at a surface = reflected + scattered + absorbed + transmitted. Light has a major impact on the photorealism of a rendered scene, but can be hard to recreate.
- Light Source** — There are several different sorts of light sources used in 3D graphics to simulate light: ambient, distant, linear, area, and spotlight. Special light effects can be recreated such as volumetric light and glow. With radiosity, an object with a high luminosity value can cast light as well.
- Lighting a Scene** — One of the ingredients of a nice rendering is realistic lighting. It is often good to use one single light source (the key light) doing most of the work, helped out by some additional, less intense lights (fill lights) which illuminate the background of the rendered object to create a smoother look. Try to avoid shadows with edges that are too crisp, since this is unusual in real life due to radiosity.
- Lighting Model** — This is a model that uses a mathematical formula to decide what will happen when light strikes an object's surface.
- Light-Map** — Luminance map generated (normally rendered) individually for each polygon and then blended with the texture map to give the impression of light and shadows falling onto the polygon without having to draw the effect on the texture itself. The advantage of separating the light-map from the texture map is that if you should want to create a new "mood" for a scene you can set up new lighting conditions for the scene, re-render the light-maps and apply them to the mesh again, without having to redraw all texture maps.
- Linear Patterning** — The repetitive placement of the active pattern cell along a line, line string, shape, arc, circle, ellipse, or curve element.



**Line-of-Sight (LoS)** — Has become quite important in modern real-time interactive simulators, especially for military purposes. To cut down on the polygon count and increase rendering performance, programmers are often forced to employ schemes to simplify terrain at large distances. This, however, has the unfortunate drawback of warping the terrain, something that may make a difference for long distance targeting purposes. Because of this, modern terrain rendering algorithms such as ROAM tend to not simplify along the primary LOS.

**Local Coordinate System** — As opposed to the world coordinate system, the Local Coordinate System is tied to a specific object. LCS are used, among other reasons, to simplify the representation of complex objects by using several, different LCSes as reference points for the object's vertices. It is also easier to transform the object if you for instance can rotate it around its own "center of gravity" instead of the origin of the World Coordinate System.

**Local Coordinates** — Every object has its own origin, which is subordinate to the world coordinate system (or other objects that are higher in the hierarchy). Local coordinates are useful for determining positions of subordinate objects.

**Local Illumination** — A mathematical model capable of creating imagery where only direct illumination is considered. Depending on the distance from the lightsource, etc, each surface in the model can be given a color and intensity. This does not include shadows, reflections and radiosity.

**Loop** — A continuous playback of an animation sequence.

**Low-Poly Modeling** — To model using as few polygons as possible, to speed up rendering and processing time. Common style for games, but as game processor engines get better, and computers faster, this is losing ground as an art form.

**LScript** — LightWave's built-in scripting language. Can be installed and used just like plugins.

**Lumel** — Short for LUMinance ELEment, the lumel is a pixel in a lightmap which constitutes the color level in a specific area of the texture map it is superimposed upon.

**Luminance** — The black and white information (brightness, sharpness, and contrast) encoded in a color. The amount of luminance contained in a color is directly proportional to the amount of light intensity.

**Luminance Map** — An image to control the luminance of a surface. Bright values in the image indicate more light intensity, dark values less.

**Luminosity** — Much like glow, luminosity is a measure of how much light a surface gives off before any light strikes it. This effect can be used to create an object that gives off its own light.

**Magnet** — This tool allows you to move points in an object as if he or she was using a magnet. It has an area of falloff where the strength of the magnet decreases gradually to 0 giving a soft selection effect.

**Map** — An attribute that can be added to an object's surface to give it a certain look. Projecting an image so that it covers the surface of an object or images that affect the way an object looks. There are a variety of different maps used to create specific effects: diffuse maps, bump maps, opacity maps, etc. Maps can be divided into bitmap-dependent texture maps and procedural maps. The latter categories can, in turn be divided into 2D maps and 3D maps.

**Mapping** — Process of making one image conform to the size, shape, and/or texture of another.

**Material** — Even if it is hidden beneath another texture map, there is an underlying material in any surfacing. The material is applied to the whole object, and can be made to look like wood, plastic, glass, metal etc. (hence the name), by modifying its properties. Also referred to as Surface.

**Material Properties** — The different properties of a material such as the ambient component, diffuse component and specular component in the reflection-illumination model.

**Matrix** — Matrices form the core of linear algebra and are important tools in most engineering disciplines. In essence a two-dimensional array of numbers, matrices are often used in transforms of different properties, such as rotation, scaling, translation, deformation, and much more.

**Memory Swapping** — The transferring of data back and forth between active RAM memory and disk. When this happens, it can considerably slow down computing tasks such as rendering.

**Mesh** — Object made up from a number of triangular faces. Also, slang used to refer to objects.

**Mesh Complexity** — Describes the amount of information (number of vertices, normals, triangles etc) used to create objects. More complex meshes need more memory and are slower to process

**Meta-primitive** — A Metaball, Metaedge or Metaface object.

**Metaform** — Option used with the Subdivide tools. It does not simply divide the individual polygons of an object, but rather renders the edges of the polygons to be smooth, making the object seem less faceted and cleaner.

**MIP-Mapping** — Using a pyramid structure of a predefined fixed amount of differently sized bitmaps (original size, original size/2, original size/4, etc) to speed up rendering time by using less detailed textures for distant objects (represented by only a few pixels on the screen), and the full-sized version of the bitmap when the objects are closer to the observer. This way, moiré-pattern can be avoided.

**Mirror** — Creates an exact mirror image of the selected object. This tool is very useful for any symmetrical object, including faces, cars, and airplanes. This tool literally cuts the Modeling time of this sort of object in half.





- Modal/Non-modal** — A modal panel must be closed before you can continue working with the rest of the application. A non-modal panel lets you shift the focus between it and another part of the application without having to close the panel—you can continue to work elsewhere in the current application while the panel is displayed. Modeler's Numeric Panel is non-modal because you can do other things while it is open. In contrast, Modeler's Display Options Panel is modal because you must close it before you can continue working.
- Modeling** — The process of creating, or recreating, an object inside your 3D software.
- Moiré Pattern** — Optical pattern created due to aliasing. Usually appears as a swirling pattern along a distant edge.
- Morgue** — A special directory, used by some studios, to hold already modelled organic body parts for other modellers to draw from. If you have modelled a good head, hands, ears, feet, etc. there is no reason to model them again.
- Motion Blur** — The blurring of objects that move while the camera shutter is open, creating the illusion of movement. Motion blur also prevents strobing caused by too-rapid movement.
- Motion Capture** — Method used to input live movements into a computer from an external source.
- Motion Channel** — Generally the same as Animation Channel, but refers only to position, rotation, and scale (i.e., not light intensity.).
- Motion Path** — The line an object follows while in motion.
- Multiplex** — The process of taking a right and left image and combining them with a multiplexing software tool or with a multiplexer to make one stereo 3D image.
- Multi-Texturing** — Applying two (or more) textures on the same face. For example, a polygon can have a texture map resembling a brick wall and then be multi-textured with a light-map to give the illusion of being lit.
- Natural Light** — Light that exists in nature, such as sunlight or moonlight, depends on the time of day, season and location on the Earth. The sunlight on a clear day has an RGB value of about R:250 G:255 B:175. For simulating overcast it might be a good idea to add the blue component, whereas a sunset could be a little more orange. As opposed to artificial light, the natural light has only one source (the sun) and can most effectively be recreated using a distant light.
- Node** — The basic graph element used to represent distinct items (vertices, faces, etc.). A signal coordinate in a grid, or finite element grid point used to describe the structure. A node will lie on each vertex of a finite element, and additional nodes may lie along element edges to define curved element topology.
- Non-Planar** — Generally refers to a polygon where all points do not reside in the same plane and can occur only with polygons using more than three points. Non-planar polygons can cause erratic rendering errors. As an example, a square piece of cardboard sitting upon a tabletop will become non-planar on all vertices when lifted by a corner. Inherent in manipulation and deformation of a model, non-planar "holes" can appear in the surface consistency of models. Solutions include "tripling" (halving the quads diagonally) or tessellating the polygons into triangles. As an example, a triangular piece of cardboard sitting upon a tabletop will remain planar on one vertex when lifted by any corner. Thus, when joined on their vertices, a group of triangles are more robust when deformed.
- Normal** — A polygon normal is the imaginary line projecting out perpendicular to a surface at any point indicating the direction of the polygon. A polygon surface normal is represented as dashed lines on selected polygons in Modeler. LightWave sees polygons or faces of an object only from the surface normal side. A single-sided polygon (like a piece of paper) with its normal facing away from the camera will be invisible to the camera from that viewpoint (unless the surface is using the Double Sided option). A vertex normal's direction is the average of the polygon normals it is connected to.
- NTSC** — National Television Standard Committee. The most common video standard in the United States and Japan. It has a frame-rate of roughly 30 fps. 60 times per second every other scan line is changed, resulting in smoother transitions. Its pixel resolution is 720x486 with a pixel aspect of .9
- Null** — Non-renderable helper-object used in modeling programs to simplify the manipulation of 3D-objects and texture mapping.
- NURBS** — Abbreviation for Non-Uniform Rational B-Splines.
- Nurnies** — American slang. See greeblies.
- Object** — A model or construction that when placed in a scene will render what it represents from the real world. An object is composed of points and faces. Points connected together to form a polygon define a face. Faces joined together form an object.
- Object Oriented Graphics** — Different from bitmap format, this image type consists of objects that have definite mathematical formulas behind them. These images always print at the maximum quality specified by the printer, unlike bitmapped images that always print at the same quality level. They can also be referred to as "vector graphics".
- Omni-Directional Light** — Same as a point light.
- Opacity** — The opposite of transparency.
- Opacity Map (or Transparency Map)** — Makes the surface more or less transparent depending on the pixel intensity (color value) of the opacity map where normally black is transparent and white is opaque.
- OpenGL** — A 3D graphics API that includes capabilities for 2D imaging. Basically, OpenGL is a set of instructions that can be used by a program to interpret images and display them on the screen. LightWave uses OpenGL for all its displays.



**Optical Light Effect** — If the observer (or camera) looks directly at a bright light source, it may appear to glow. If the light is refracted through a lens or even your own eyelashes (try squinting towards a spotlight!), the light will appear to form star-like patterns.

**Orbit** — To travel around a target - more commonly circular, but a comet's orbit can be elliptical.

**Origin** — The world Origin is the absolute center of the LightWave universe. A local Origin is the center of an object. Both are defined by the XYZ coordinates of 0, 0, 0.

**Orthogonal** — A view that displays a parallel projection along one of the major axes. In an orthogonal view, the camera is oriented to be perpendicular (orthogonal) to specific planes: the Top view faces the XZ plane, the Front view faces the XY plane, and the Right view faces the YZ plane. An orthogonal view eliminates the effect of distance from a viewpoint, which provides a useful means of locating points and objects in 3D space and is particularly helpful when modeling objects in wireframe mode. An orthogonal view is in contrast to a perspective view.

**Orthogonal Direction** — There are six different orthogonal directions in a three-dimensional space: up, down, back, forward, left and right.

**Orthographic Projection** — Viewing system where the projectors are parallel and therefore don't create a perspective with foreshortening.

**PAL** — Phase Alternating Line. The most common video standard in Europe. It has a frame-rate of 25 fps. It is interlaced, which means that 50 times per second every other scan line is changed, resulting in smoother transitions. The resolution is 720x576 pixels and the pixel aspect ratio is 1.0667.

**Pan** — To rotate the camera horizontally. As opposed to the orbit movement, pan rotates the camera around a single axis, as if it were mounted on a tripod.

**Panel** — In a 3D program, a screen that serves many functions such as informing you of errors, asking for user input, or informing you of the state a program is currently in. Otherwise known as a dialog, window or requester.

**Parabola** — A plane curve formed by the intersection of a right circular cone and a plane parallel to an element of the cone or by the locus of points equidistant from a fixed line and a fixed point not on the line.

**Parametrisation** — Technique for assigning values to the edit points as they are spaced along the length of a curve. Can be either uniform parametrisation or non-uniform parametrisation (chord length). The first edit point on the curve has the value 0.0 (regardless of whether it is uniform or non-uniform) and the following edit points are assigned greater values the closer they lie to the other end.

**Parameters** — Also generally known as properties, parameters are the "atomic" elements of a property set whose values determine the behavior of something. A parameter is one degree of freedom. You can set parameters in property editors.

**Parent** — An object that influences the motion of another object in a hierarchy, called the "child".

**Parenting** — The process of creating a hierarchical organization of objects in a scene. In parenting, an object (called the parent object) is "parented" to another object (called the child object). Parenting relationships can be nested to any degree, so that one or more objects are the children of another object, which is in turn the child of another.

**Particles** — 2-dimensional objects typically used in large quantities to create effects like rain and explosions.

**Passive Polarised 3D glasses** — 3D glasses made with polarising filters. Used in conjunction with a view screen that preserves polarised light.

**Penumbra** — A partial shadow, as in an eclipse, between regions of complete shadow and complete illumination, a fringe region of partial shadow around an umbra.

**Perspective** — A traditional art method of creating the illusion of three-dimensional form and distance on a two-dimensional surface. Perspective provides a three-dimensional view of the scene that indicates depth. In a perspective view, objects appear to converge toward a central vanishing point, and objects closer to the camera appear larger than those farther away. A perspective view is in contrast to an orthogonal view.

**Perspective Projection** — Simulating three-dimensionality by using foreshortening that triggers the human perception to interpret a two-dimensional image as if it was three-dimensional. An object is drawn smaller and smaller the further it is from the observer. This is achieved by using a center of projection to which all projectors converge, as opposed to where the projectors are parallel.

**Phong Shading** — The most frequently used interpolative shading technique used today. It uses a linear combination of three components - the ambient component, the diffuse component and the specular component. The placement of the highlight is less dependent on the underlying polygons as Gouraud shading since Phong shading interpolates the normals on a per-pixel basis instead of interpolating the light intensity based on the distance to the three vertices.

**Photorealism** — The process of generating computer images that mimic photographs.

**Pitch** — The amount that the camera or an object in the scene is tilted up or down. If you nod your head "yes", you are rotating your head in the pitch axis.

**Pivot Point** — A single point, usually in the geo-center of an object that is used for many functions. It is the point that is addressed to locate an object's position in 3D space. It is also the point around which all rotational moves are made and the reference point for transformations and scaling.

**Pixel** — Short for Picture Element, the smallest element of computer or video display.

**Plane** — Refers to a two-dimensional (i.e., flat and level) surface. Imagine a plane as a piece of glass that is infinitely large, but has no depth.

**Plugin** — A program that works with and extends the functionality of LightWave.





- Point** — A fundamental building element of an object in 3D space with an XYZ location. Point coordinates are the minimum information from which the geometry of an object can be calculated.
- Point Light** — Light source emitting light in all directions (omni-directionally) from a single point in space, think “light bulb”. It takes six shadow calculations (one in each orthogonal direction) to render shadows generated by a point light, which means that inserting multiple point lights into a scene might slow down rendering time considerably.
- Polygon** — Geometric shape in one or many planes. Polygonal Modeling consists of using many faces to create the shape. Since polygons in most cases are faceted simplifications of a much smoother shape, they are more or less inaccurate, as opposed to the more organic NURBS. The more the tessellation, the higher and the closer the accuracy compared to the desired shape.
- Poly-Line** — A geometric entity composed of one or more connected segments that are treated as a single entity.
- POV** — Abbreviation for Point of View.
- Primary Colors** — There are three primary colors of light: red, green and blue (RGB). Light colors are additive, which means that if these three colors are combined equally, the result is a white light. Black is thus the absence of light.
- Primitive** — Basic geometric shape used in Modeling. Some primitives consist of a combination of different primitives. Cone, box, sphere, tube, torus, and disc are common primitives.
- Procedural Map** — A map (often three-dimensional) generated mathematically using a set of user-customised variables instead of using an image. The procedural map does not need texture coordinates.
- Procedural Textures** — Mathematically generated textures (2D and 3D). Their advantage is that they are largely independent of the projection type.
- Projected shadow** — A shadow that falls from an object and projects on a surface.
- Projection Map** — A mapping procedure that allows you to apply the map to multiple objects as if they were one.
- Quad** — A polygon with four sides, short for quadrilateral.
- Quantise** — This tool causes points to snap to a specific (X, Y, Z) coordinate. This tool is generally used when a lot of precision is required.
- Quaternion** — Quaternions are mathematical objects consisting of a scalar and a vector which together form a four-dimensional vector space. Although having interesting uses in mathematics, their main use in computer graphics resides in their capability of easily representing 3D rotations. Although impossible to visualise, they suffer from no singularities like Euler angles, and are also easy to smoothly interpolate for keyframe animation (using a mathematical operation called SLERP for Spherical LinEar INTERPolation).
- Radiosity** — A more physically correct approach (developed in 1984 by Siegel and Howell) to simulate propagation of light in a virtual environment. It takes into account the fact that light bounces off a surface and creates diffused lighting on the surrounding objects. The scene is divided into a certain amount of triangles that are used to represent the original scene (which speeds up the time-consuming process), and then light interaction is calculated using these triangles. As far as visual quality is concerned, the more crucial the part of the scene the denser the triangles must be. This technique creates much more realistically lit environments, however it takes much longer to render due to the massive amount of calculations.
- Rail Clone** — This tool creates multiple copies of an object that are evenly spaced along one or more user-defined curves.
- Rail Extrude** — Used to extrude polygons along a specified line or combination of lines. This allows you to create a shape other than that created from a normal, linear extrude.
- Rapid Prototyping** — The process by which a computer model of a 3D object is turned into a real object. Rapid prototyping methods vary but often involve laying down strata of base material which is then bonded together using a substance like cyanoacrylate (superglue).
- Rasterisation** — The process of, on a per pixel basis, determining what value to assign to the pixels on the screen from a vector-based image.
- Ray Traced Shadow** — Shadow created by tracing the light rays from a light source. The ray traced shadows are more accurate than those created by shadow maps, but take more time to render and always have crisp edges.
- Ray Tracing** — An advanced rendering technique capable of calculating reflections, refractions and shadows. Ray Traced renderings take more time to generate, but have a more photorealistic quality than simple scanline rendering.
- Ray Tracing Depth (Ray Recursion Limit)** — Number of times the light bounces off a surface when ray tracing. Used to create reflections and/or refractions. For example, ray tracing two mirrors facing each other with the ray tracing depth set to 3 will allow the image of the reflected mirror to show the first mirror in it.
- Reflection** — Light that bounces off a surface. A mirror is highly reflective, whereas the reflection of a matte rubber surface is insignificant.
- Reflection Map** — Simulates reflections in a surface using the reflection map instead of actually ray tracing the reflected image. This speeds up rendering time, but can also be a give-away if the scene is animated.
- Reflection-Illumination Model** — Model used when creating two-dimensional images from three-dimensional meshes. To produce more realistic and convincing images, the reflection model imitates attributes of real-life objects.
- Refraction** — When light passes through a transparent material and into a denser, or less dense, medium the light rays are refracted and change direction. Each material has its own refraction and, depending on the density of the material, the refraction is more or less evident. Refractions are calculated similarly to reflections using ray tracing.



**Refraction Index** — A value describing the amount of refraction that takes place in a specific transparent material. For vacuum the refraction index is 1.0000, for air 1.0003, for glass approximately 1.5 and for water 1.3333.

**Refraction Map** — An image to control the level of refraction across a surface where dark values indicate a low refractive index and bright ones a high refractive index.

**Render** — To mathematically generate geometries, algorithms, reflections, etc. Our work would be meaningless without the ability to render. Creating a final image of a model that shows all of the surface properties which have been applied to an object. This process involves adding all colors; bump maps; shading; and other elements that add realism. In a normal 3D program, you can view the wireframe of the created image. When an image is rendered, the wireframe is covered with the specified colors and properties.

**Render Pass** — Division of a scene according to different aspects (such as highlight, mattes, or shadows) for the purposes of applying specific rendering options. Passes can then be composited during post-production. The default pass is the beauty pass, which includes all objects in the scene. Preset passes include matte, shadow, and highlight passes. You can also define your own passes to include any object you want to be affected by specific rendering properties. Render passes are further divided into partitions.

**Rendering Pipeline** — Description given to the process of creating the rendered images. Some studios have a process by which all the images go through. Some render in passes, one for the base, then the shadows, then the reflections, etc. This process is the pipeline.

**Resolution** — The number of picture elements in an image.

**Revolution** — A Modeling term defining a surface made by rotating a curve around the axis of another curve.

**RGB Color Model** — A color model that mixes the three primary colors to produce colors. To create yellow, red and green are mixed without any blue component. The higher the value of the red, green and blue, the clearer the color. Lower RGB values give darker colors, while higher RGB values give lighter colors.

**Rigging** — The process of making an object ready for animation. This does not have to be just characters; it is the same for all objects. Rigging involves creation and implementation of bones, hierarchies, clamps, weight maps and sliders.

**Right-Handed Coordinate System** — A coordinate system (frequently used in 3D-graphics applications) whose positive Z-axis emerges from the screen towards you, just like the one used in mathematics, as opposed to the left-handed coordinate system.

**Roll** — The amount that a camera is tilted to the left or right. Also known as the Bank Angle.

**Rotoscoping** — A technique in which video or film images are placed in the background of a scene, one frame at a time. You can use these reference images to create your own animation by tracing objects from the images or matching your objects with the images' motion. You can zoom and pan the scene while maintaining a perfect registration with the imported background.

**Row Interleaved** — A format to create 3D video or images in which each row or line of video alternates between the left eye and the right eye (from top to bottom).

**Rule-Based Design** — The definition of relationships between objects in the design. Another name used to describe Knowledge-Based Design.

**Scalar** — A quantity, such as mass, length, or speed, that is completely specified by its magnitude and has no direction, a one dimensional value.

**Scanner** — Device for reading images (from books, photos etc.) into the computer. This is useful for creating realistic textures. With a 3D scanner it is even possible to capture three-dimensional objects and convert them into models.

**Scene** — A Scene is a LightWave project defining the objects loaded and their motions, the number of lights and their values/motions, the resolution of the final image, special effects, Camera settings, and so on. This ASCII text file is generally saved from Layout.

**Scrub** — The process of manually dragging the frame advance control slider on the timeline to see or hear its effect on video/audio.

**S-Drill** — Refers to Solid Drill. Acts just as a drill would, using a 3D object as the drill bit. This tool can be used to take sections out of objects or perform other functions that a drill might perform.

**Seamless** — A seamless texture can be tiled without visible transitions where the bitmap begins and ends. This means that the upper part of the bitmap can be placed next to the lower part of the bitmap, or the right can be placed next to the left, forming a pattern that is apparently coherent.

**SECAM** — Séquentiel Couleur à Mémoire. The television broadcast standard for France, the former USSR, and various eastern European countries. Like PAL, SECAM is based on a 50 Hz power system, but it uses a different encoding process and displays 819 lines interlaced at 50 fields per second. SECAM is not compatible with NTSC or PAL, but conversion between the standards is possible.

**Secondary Animation** — Once the main movements of animation have been applied, this refers to the detail animation step. Hoses bouncing when a robot walks and flab wiggling when a heavyset character moves are examples of secondary animation.

**Sector** — Convex volume used to speed up rendering time.

**Self-Shadow** — Object property that allows one part of a complex object to cast a shadow onto another part of that same object (example: the branches of a tree casting shadows onto its trunk).



- Self-Illumination (or Luminosity)** — Allows non-homogeneous self-illumination of the surface. Some parts can be self-illuminated, some partially self-illuminated, and some not at all, based on the pixel intensity of the self-illumination map (normally black=left unchanged, white=self-illuminated).
- Session** — A session is a single use of an application. A session begins when you first start the application and ends when you exit.
- Shaded Mode** — Shaded mode generally refers to a viewport that has its Rendering Style (Display Options panel or viewport title bar) set to something other than wireframe. These modes show polygon surfaces with some level of shading.
- Shading** — Simulating that an object is lit by a light source.
- Shadow** — An area that is not or is only partially lit because of the interception of light by an opaque object between the area and the light source.
- Shadow Map** — Bitmap generated by the rendering engine during a pre-render pass of the lit scene. Generally a shadow map is less precise than a raytraced shadow, but takes less time to render. As opposed to a ray-traced shadow, a shadow map can create shadows with smooth edges. Furthermore, the shadow map is unable to show the color cast by a transparent object. The quality of the shadows in the rendered image depends on the size of the shadow map. The bigger the map the nicer the shadows. A shadow map that is too small might result in aliased or stairstepped edges. For example, a 256x256 shadow map (65k) is normally sufficient for resolutions of 320x200 and less. If an object is far away from the light source, the shadow map will have to be increased in order to maintain visual quality. If the final rendering is in high-resolution, the shadow map also needs to be hi-res.
- Skew** — Modifying an object by tilting it.
- Skin** — Creating three-dimensional object from two or more two-dimensional shapes and then extruding them along a path.
- Smoothing** — Technique that, when rendering or shading, smoothes out the edges between segments making objects appear smoother than their geometry really is.
- Soft Shadow** — does not have hard edges. Traditionally, ray traced shadows always have hard, crisp edges, whereas shadow-mapped are considered soft shadows. With global illumination methods, physically accurate, soft-edged shadows are achievable at a cost in rendering time.
- Space** — A set of elements or points satisfying specified geometric postulates: non-Euclidean space. The infinite extension of the three-dimensional region in which all matter exists.
- Specular** — This property determines how shiny (and sometimes wet) an object appears. It represents the highlight that the light creates when shining on an object.
- Specular Component** — Part of the reflection-illumination model. Specular surfaces are capable of reflecting light like a mirror.
- Specular Map** — Replaces the specular component of the reflection-illumination model, thus only visible in an object's surface's highlights.
- Specular Reflection** — The brightest area on a surface, reflecting surrounding light sources, creating highlights.
- Spherical Image Map** — One of the many methods of applying a texture to a surface. This method applies the texture to an object as it would apply the texture to a sphere. There are many other methods of texturing objects, such as Cubic and Planar image mapping.
- Spinning Light Trick** — Trick to create soft shadows with ray tracing. It involves parenting multiple lights to a null and spinning the null.
- Spline (Curves)** — Layout uses splines or curved paths between keys while moving items about. When Modeling, splines refer to open or closed curves.
- Spline Cage** — A spline cage is usually a three-dimensional object made up of connected spline curves.
- Spline Patching** — The process of adding polygons to fill in areas outlined by splines.
- Spot** — A small opaque circle placed in front of a light, usually to remove the specular hot spot from an object.
- Spotlight** — A lightsource emanating light in one direction only, in the shape of a cone.
- Staircasing (or Stairstepping)** — A graphical flaw caused by insufficient resolution. When rendering an object its contours might stand out too crisply from the background and the pixels might be obviously "zig-zagged", or look like stairs. To prevent this, pixels can be blended into their neighbours' colors by antialiasing.
- Stencil** — When using the drill tool, the stencil option adds the details of the drilling polygon to the polygon being drilled. This creates new geometry on a shape.
- Stereoscopic 3D** — Two separate photographs taken from slightly different angles that, when compiled, appear three-dimensional.
- Stretch Tool** — Allows you to change the size of an object along a particular axis.
- Subdivide** — Divides any selected polygons with three or four sides into smaller polygons. This makes an object appear smoother, but also makes the model more complex.
- Subdivision Surfaces** — Subdivision surfaces are a technique to create a smooth curved surface from a coarse polygon mesh. Several different subdivision schemes have been developed since Ed Catmull and Jim Clark first introduced the idea back in 1978. The most well known schemes are the triangle-based Loop scheme and Catmull & Clark's original scheme, which is based on quad polygons. Subdivision surfaces were introduced to the public in the Pixar movies, *Toy Story 2* and *Gerl's Game*.
- SubPatch** — Refers to a Modeling mode wherein polygons become a cage that controls an underlying mesh of subdivision surfaces.



**Subtractive Opacity** — Type of opacity that subtracts the background color from the transparent object's material color.

**Super Pixel** — They are created in a supersampling image. Groups of the super pixels are filtered into the one single pixel that is displayed on the output display.

**Supersampling** — Generating images at a resolution  $n$  times  $n$  larger than the display resolution and then filtering the so-called super pixels into the smaller resolution image, creating smooth images with antialiasing.

**Surface** — Essentially, the surface is the skin of an object. The surface attributes can be changed using the Surface panel. Many features, such as the name and color attributes, affect the appearance of an object. A single object can have multiple surface names, each with its own independent attributes (e.g., color), and multiple objects can share the same surface name(s).

**Tangent** — A straight line that makes contact with a single point along a curve.

**Taper** — Modifying an object by progressively narrowing it along an axis.

**Target** — In aiming the camera, the target is the object that is selected for the camera to point toward. The target is kept in the center of the camera's view.

**TD** — Technical Director. A job in a studio that mainly concerns making rigs, and to help out the other departments wherever possible. They are the problem solvers.

**Tessellation** — Increasing the detail level of a polygonal 3D model by increasing its number of polygons, usually triangles. The more triangles, the smoother the shape and subsequently the larger the model. The tessellation can be performed by dividing one triangle into two (or more) smaller ones. By doing this the new, more faceted model can be modified without losing too much of its smoothness.

**Texture** — Normally texture describes the attributes of a surface, for example if it's coarse, smooth, wrinkled or rough, but it is also used with the meaning of texture map. There are textures made from bitmaps (texture map), and textures generated mathematically (procedural map). Textures specify how the surface of an object will look, and can be anything from simple, solid colors to complex images representing the surface of the object. The simplest example of a texture is placing a picture on a flat plane. The picture is the texture being applied to the plane.

**Texture Coordinates** — Coordinates used to describe how to map a texture map onto an object. There are different kinds of techniques to apply the texture: planar, cylindrical, spherical, cubic, front, and UV. Their names indicate how the texture is projected onto the object the mapping coordinates are applied to. Procedural maps do not need texture coordinates.

**Texture Map** — Map wrapped over the surface of an object. The texture map needs to be spaced correctly in U and V direction over the object.

**Texture Mapping** — The process of projecting a (usually) two-dimensional image onto a three-dimensional face such as a triangle or a quad, texture mapping is a relatively cheap way of adding tremendous detail to a scene without resorting to extremely detailed meshes that take an inordinate amount of memory and time to render.

**Tiling** — Repeatedly placing the same texture next to itself on the same surface, creating a pattern from one image. This is achieved by increasing the texture coordinates on a polygon to a value greater than 1. Normally, the entire bitmap is tiled from 0.0 to 1.0 in  $u$ - ( $=x$ ) and  $v$  ( $=y$ ).

**Timeline** — The slider below the Layout viewport representing time in animation.

**Transformation** — The act or an instance of transforming. The state of being transformed. A marked change, as in appearance or character, usually, /hopefully/ for the better.

**Truck** — To move the camera in the viewing plane.

**Twist** — Modifying a mesh by rotating its vertices non-uniformly along an axis.

**U-Direction** — Represents a grid line in one direction (normally that of the original curve) of a UV texture map.

**Umbra** — A dark area, especially the blackest part of a shadow from which all light is cut off. The completely dark portion of the shadow cast by the earth, moon, or other body during an eclipse.

**Unify** — This command creates single-sided polygons according to the properties of their surface normals. Basically, this tool transforms polygons that share points into a single polygon.

**Union** — One of the options in the Boolean tool. This option makes an object that is a combination of the two objects.

**UV-grid** — A grid system for identifying points on a surface. The U-direction and V-direction are for the surface, what the X-axis and Y-axis are for the coordinate system.

**V Map** — V Map is an abbreviation for vertex maps. V Map provide additional information associated with object points (vertices), like weight, UV and morph maps.

**V-Direction** — Represents a grid line in one direction (normally "up-down") on the surface of an object.

**Vector** — Entity with both magnitude and direction. A three-dimensional vector is written as:  $V=(v1, v2, v3)$  where each component is a scalar.

**Vertex** — (pl. vertices) Point at which the sides of a polygon intersect. In 3D graphics, vertex may also refer to a point at a specified location in three-dimensional space, and such a point is the smallest component in a 3D mesh.

**Vertex Count** — The number of vertices in a scene. Remember, the higher the mesh complexity the longer the rendering time.





**Vertex Normal** — Even though it is a single point in three dimensional space, its normal can be calculated based on the normal of the face they are describing. The three vertex normals of a single triangle without any neighboring triangles are set to be the same as the polygon's normal. For triangles surrounded by other triangles, the vertex normals are the average of the surrounding face normals.

**View Frustum** — Representing the field of view of the camera, the view frustum is a pyramid volume with the top sheared off. The top of the pyramid represents the viewport of the camera (usually the screen), and is often called the near (or hither) plane, while the bottom is called the far (or yon) plane.

**View Frustum Culling** — Removing faces that lie outside the observer's view. Only the face that is within the view frustum is kept for rendering — speeding up rendering time and helping to maintain a high framerate.

**Viewport** — Window area displaying orthogonal or perspective projection in a 3D application. The screen can either contain one big viewport or several smaller, tiled viewports. By simultaneously using several viewports displaying a three-dimensional object from different sides (e.g. top, front, left, perspective), modeling in a virtual 3D environment is made possible.

**VIPER** — Versatile Interactive Preview Render. A window that provides you with an interactive previewing system.

**Volume** — When selecting, a volume of an object is a 3D representation of the area to be edited. When editing, all of the parts of objects contained within this 3D selection can be edited without changing what lies outside of the selection.

**Volumetric Fog** — Fog that, opposed to ordinary fog, is restricted to fit within a containing volume.

**Volumetric Light** — Light simulating illumination of particles floating in mid-air, thereby making the light cone itself visible.

**Vortex** — A tool that rotates an object more in the center than in the outer edge. This tool can be easily related to a tornado, where the wind in the center moves faster than the wind in the outer part of the cone.

**Voxel** — Short for VOLUME ELEMENT, this term refers to a specific rendering technique common in medical visualisation as well as some interactive media. In essence, a voxel is a three-dimensional pixel, that is, a cube, with a specific color.

**Weights** — The strength of influence on a particular vertex of an assigned deformer, such as a bone. See V Maps.

**Weld** — This command takes the selected points and combines them into one point, a single point that is specified by the last point that is selected.

**WIP** — Short for Work In Progress.

**Wireframe** — A way of visualising geometry by drawing lines between its vertices but not shading the surfaces within.

**World Coordinate System** — The coordinate system, normally in three dimensions, used to describe the location in space of a specific point called vertex.

**X-Axis** — Usually is the axis that is left-right/side-side.

**Yaw** — To turn about the vertical axis, also known as heading.

**Y-Axis** — Usually is the axis that is up-down/top-bottom.

**Y-up** — Coordinate system with the Y-axis pointing upwards.

**Z-Axis** — Usually is the axis that is in-out/front-back.

**Z-Buffer** — Also called depth buffer, the z-buffer is a two-dimensional matrix of 16 or 32-bit integers with the same dimensions as the screen (or viewport). Whenever a polygon is drawn to the screen, the rasteriser checks the corresponding z-buffer value for each screen coordinate and skips drawing the current pixel if the z value is marked as being closer. This allows for some nice effects such as overlapping 3D models, and completely solves the rendering-order problem.

However, this comes at the price of slower performance and greater memory usage, two factors that have become more or less moot with the proliferation of modern 3D accelerators that tend to support z-buffers in hardware.

**Z-Up** — Coordinate system with the Z-axis pointing upwards.







# Index

---



# Index

## A

Advanced Tab 36  
 AH\_CelShader 38  
 Alpha Channel 270  
 Anaglyph Stereo\  
   Compose 300  
   Simulate 300  
 Animated Dither 297  
 Aura 2.5 Export 299

## B

Basic Surface Parameters 9  
 Batch Render on One Computer 293  
 BESM 38  
 Bloom 300  
 Blotch 39  
 BRDF 39  
 Bump Map 11  
 Bumprgb 39

## C

Camera Basics  
   Frame Aspect Ratio 277  
   limited region 277  
   Masking out a region 291  
   Memory 277  
   Motion Blur Effects 281  
   Pixel Aspect Ratio 276  
   Resolution 276  
   Segment memory limit 277  
 Caustics 287  
 Checker 42  
 Chroma Depth 301  
 Color 9  
 ColorCube 42  
 Color Saturation 298  
 Compositing 295  
   Alpha Images 296  
   Background Image 295  
   Foreground Fader Alpha 297  
   Foreground Images 296  
   Foreground Key 297  
 Corona 302

## D

Depth-Of-Field Blur 304  
 Diffuse 9  
 Digital Confusion 303  
 Distributed Rendering\  
   Introduction. *See* ScreamerNet  
 Dither Intensity 297

## E

Edge\_Transparency 43  
 Editing Tab 271  
 Envelopes and Textures 12  
 Environment Tab 37  
 Exposer 304  
 Extended RLA Export 304  
 Extended RPF Export 304

## F

Fast Fresnel 44  
 Field Stereo 305  
 File Saving 295  
 Flare2Alpha 300  
 Frame Rate 270  
 Full Precision Blur 305  
 Full Precision Gamma 305

## G

Glow  
   Glow Settings 298  
 gMIL 51  
 Gradient  
   Changing Key Positions 36  
   Changing Key Values 36  
   Gradient Bar 35  
   Smoothing Between Keys 36

## H

Halftone 44, 298  
 HDR Exposure 305  
 High Quality Blur 311  
 HSLColorCube 43

## I

Image Editor 268  
   Getting to 268  
   Source Tab 270  
   Using the, 269  
 Image Map Projection 19  
   Cubic Projection 21  
   Cylindrical Projection 20  
   Fixed Projection 23  
   Front Projection 21  
   Planar Projection 19  
   Spherical Projection 20  
   UVs and Projection 23  
 Image Sequence Settings 271  
 Image Viewer 294  
 Interference 45  
 Interlace 270



## L

Layer Type\  
     Gradient 34  
     Image Mapping 19  
     Procedural Texture 24  
 Light Properties  
     Global Illumination:Lighting Considerations 285  
     Global Illumination:Radiosity 282  
     Global Illumination:Radiosity and High Dynamic Range  
         Images 286  
     Global Illumination:Radiosity Settings 284  
     Global Illumination:Shading Noise Reduction 280  
     Global Illumination:Using Radiosity 286  
     Global Illumination:Volumetric Radiosity command 285  
 Limit Dynamic Range 297  
 Limits 311  
 LScript 45  
 LScript/RT 45  
 Luminosity 9  
 LW\_Hypervoxels 299  
 LW\_Hypervoxels\_Doubler 299  
 LW\_HyperVoxels\_Shader 44  
 LW\_Rust 45  
 LW\_Snow 46  
 LW\_Water 46

## M

Math Filter 299  
 Mirror 16  
 Morph Mixer. *See* Endomorphs  
     Morph List. *See* Endomorphs  
 Morph Targets. *See* Endomorphs

## N

Negative 314  
 Network Render 293  
 NightVision 314  
 Node Editor 54  
 Normal Color 50  
 NTSC\_Legalize 314  
 Numerical Settings 9

## O

Object Edit Mode 6  
 Object Mode 49  
 Overlapping Objects 311

## P

PAL\_Legalize 314  
 Photoshop PSD Export 306  
 Pixel Filters 298  
 Polygons  
     Double Sided 13  
 Preset Shelf 8, 356  
 Preview Window 8  
 Procedural Tex  
     Additional 31

Procedural Texture  
     Turbulence 30  
     Underwater 30  
 Procedural Textures 25  
     Brick 25  
     Bump Array 25  
     Checkerboard 26  
     Coriolis 31  
     Crumple 26  
     Crust 26  
     Cyclone 32  
     Dented 32  
     Dots 26  
     FBM 27  
     FBM Noise 32  
     Fractal Noise 27  
     Grid 28  
     Hetero Terrain 32  
     HoneyComb 28  
     Hybrid Multi-fractal 32  
     Looping Wave Ripples 29, 30  
     Marble 28  
     Multi-fractal 33  
     Puffy Clouds 33  
     Ridged Multi-fractal 33  
     Ripples 29  
     Smoky 1, 2, and 3 29  
     Turbulent Noise 33  
     Value 30  
     Veins 30  
     Wood 31  
 Processing Tab 271

## R

RealFresnel 45  
 Reference Object 18  
     Freezing Reference Object 18  
 Reflection 10  
 Render Buffer Export 310  
 Render Buffer View 301  
 Render Frame 292  
 Rendering a Limited Region 291  
     Memory Considerations 291  
 Rendering Without LightWave 293, 330  
 Render Options  
     Auto Frame Advance 274  
     Enable Viper 276  
     Monitoring Progress 275  
     Output Tab:Fader Alpha 290  
     Output Tab:Save animation 288  
     Output Tab:Saving individual images 290  
     Output Tab:Special animation types 288  
     Render Complete Notification 274  
     Render Frames 274  
     Render Tab:Data Overlay 278  
     Render Tab:Multithreading 280  
     Render Tab:Ray Recursion Limit 280  
     Render Tab:Ray Trace Reflection 279  
     Render Tab:Ray Trace Refraction 279  
     Render Tab:Ray Trace Shadows 279  
     Render Tab:Ray Trace Transparency 279  
     Viewing the finished image 275



Render Options\  
   Output Tab 288  
   Render Tab 278  
 Render OptionsLOutput Tab  
   Selecting a Filename Format 290  
 Render Scene 292  
 Render Selected Object 292  
 Render Tab 274  
 Repeat 16  
 Reset 16

## S

SasLite 299  
 Scene Edit Mode 7  
 ScreamerNet  
   Batch Rendering on One Computer 330  
   Command Directory 327  
   Drive Mapping 325  
   Organizing Configuration Files 326  
   Share and Share Alike 324  
   Show Time 328  
   Troubleshooting and limitations 330  
 Sequence Digits 270  
 Shaders 38  
   AH\_CelShader 38  
   BESM 38  
   Blotch 39  
   BRDF 39  
   Bumprgb 39  
   Checker 42  
   ColorCube 42  
   Edge\_Transparency 43  
   Fast Fresnel 44  
   Interference 45  
   LW\_HyperVoxels\_Shader 44  
   LW\_Rust 45  
   LW\_Snow 46  
   LW\_Water 46  
   Pulse 50  
   RealFresnel 45  
   Steamy\_Shader 50  
   Surf Mixer 50  
   Thin Film 50  
   Weave 51  
   Zor 51  
 Sharing Points 12  
 Skelegons. *See* Setup Tab  
 Smooth Threshold 12  
 Soften Reflections 310  
 Specularity 10  
 SpriteEdger 316  
 Steamer 299  
 Super Cel Shader 47  
 Super Cel-Shader  
   Brightness 48  
   Colored Lights 48  
   Defining Color Zones 47  
   Specular Highlights 48  
 Surface Baker 48  
   Baking Tips 48  
   Image Mode 49  
   Instant Radiosity 49

Surface Edit Modes 6  
 Surface Editor 4  
   Introduction 4  
   Mass Surface Changes 7  
   Object Edit Mode 6  
   Scene Edit Mode 7  
   Surface List 7  
   Surface Management 5  
   Surfacing Ideas 4  
   Surfacing Objects 4  
 Surface Editor Panel 5  
 Surface List 7  
 Surface Size, Position, and Rotation 17  
 Surface Smoothing 12  
 Surf Mixer 50

## T

Texture Color and Texture Value 25  
 Texture Editor 14  
   Blending Layers 14  
   Copy and Paste 14  
   Image Properties 15  
   Layer Order 14  
   Texture Layers 14  
 Texture Filter 310  
 Texture Placement 17  
 Texture Scale 25  
 Thin Film 50  
 Translucency 11  
 Transparency 11

## U

Using Separation 13  
 UV Texture Maps 23

## V

Vector Blur 311  
 Video Legalize 312  
 Video Tap 312  
 Vignette 316  
 VIPER 8  
   Print Assistant 292  
 Virtual Darkroom 313

## W

WaterMark 316  
 Wave Filter Image 317  
 World Coordinates 18

## Z

Z Shader 50