

Introduction to Low-Polygon Game Asset Texturing

With NewTek Lightwave3D, Adobe Flash, and Adobe Director

Overview

This document describes some basic methods for texturing low-polygon 3D geometry for use in Shockwave3D games using NewTek Lightwave and Adobe Director. It is intended for the novice Lightwave and Director user, but there are tips and tricks sprinkled throughout that may help the more experienced developer.

To follow along, you will need a copy of Lightwave3D installed. You do not need Director for this tutorial, but it does assume that the content is destined for a Shockwave3D environment. Most of the content applies to any sort of low-polygon texturing, however.

An Introduction to Texturing

Texturing is the process of applying information to a model's geometry to tell the game engine how to draw the model on the screen. As in the real world, this is determined exclusively by defining how the object reacts to light. As such, the process of texturing a model is the process of defining the various characteristics of the object as it relates to light.

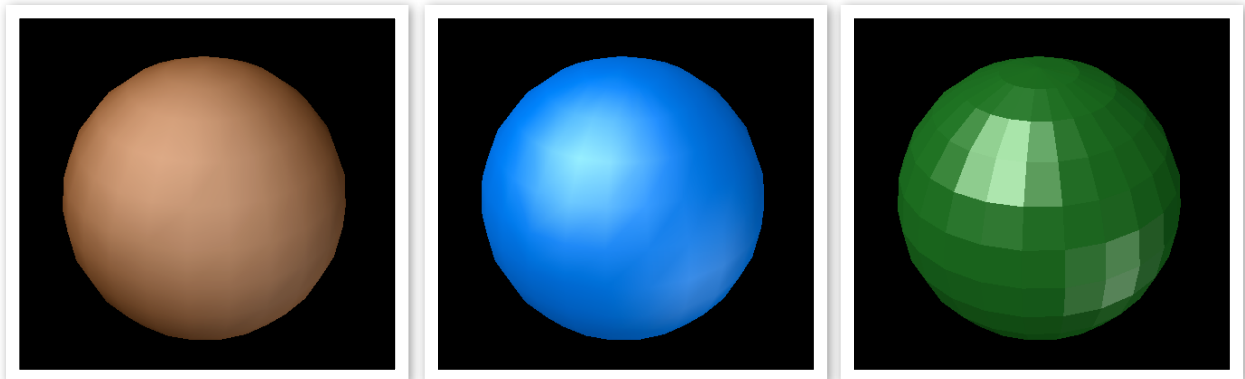
For instance, “reflectivity” is a property of a surface that determines how it reacts to light. If light bounces off of it unaffected, so that the light ray coming in is the same as it is coming off of the surface, then the surface is highly reflective.

The light-reactive properties of objects may be defined in one of two ways. You can either have a global “material” setting for the property of an object, which basically states that the entire object should have the same value for that property. Or, you can define that property using an image, which allows you to have different values for the property across the surface of the object.

For instance, think of a blob of mercury versus a tarnished mirror. The blob of mercury is going to be reflective across its entire surface, so it is a good candidate for what we will call “material-based” texturing of the reflectivity property. But a tarnished mirror will have different amounts of reflectivity across its surface. You will want precise control over where it is reflective and where it isn't, and by using an image to draw the scratches and splotches, you can achieve that in an artistic fashion. In this case, you will want to use image-based texturing for the reflectivity property.

Material-based texturing

Below are some samples of different effects you can achieve with the simple material settings:



Each of these objects use no images at all in their texture definitions, and are exactly the same except for having different global color, luminosity, specular, glossiness, and smoothness settings for their textures (more about these settings later).

Material-based texturing is much easier than image-based texturing, because there's a lot less work and precision involved: you don't need to create image maps, and you don't need to define how those images wrap around the object. You just provide one setting for that property, and it applies to the entire surface uniformly.

Most of the objects you create for a game will need to use at least one image, but you only have to use image maps for the aspects of your object that aren't uniform. For instance, you will often use an image to define the *color* of the object, but use a simple setting for the *reflectivity* (say, zero). Or you might give an object a uniform color, but use an image map for the specularly (highlight).

The more properties of objects you can get away with defining with “material” properties – that is, without using an image – the more texture memory you save on the video card, and thus, the more textures you can use in other places. Whenever possible, define the properties of an object with material settings.

Materials and meshes in Shockwave3D

There is one note of caution here. You might be tempted, then, to break up a model into pieces that can be handled with material-based texturing and parts which can't. There's a balancing act there. Each separate texture you use in an object becomes a separate *mesh* in the Shockwave3D engine, and one of the performance-draining elements of the Shockwave3D engine is the total number of meshes. This is because it has to push out a new texture definition for each mesh.

Because of this, you want to use the fewest number of textures as possible to make your model look good. If you're going to be using an image to define color for an object anyway, try to use a single image that covers the entire model, rather than six images that cover different parts of the model, or one image and six different material settings.

Image-based texturing

Very few objects have the same light-reactive properties uniformly over their surfaces, so we apply 2D textures we create in applications like Flash or Photoshop to our 3D models to assign those properties.

This raises a problem, though. A rectangular 2D image will not naturally conform to most 3D shapes you will create for a Shockwave3D game. Some objects can accept 2D shapes nicely, such as wrapping an image around a cylinder or projecting it onto a plane. But an archway of rock or a pirate ship may have lots of nooks and crannies that have no natural way of applying a 2D image to.

To get around this problem, you must tell Lightwave where the polygons are attached to the 2D image by “unwrapping” or “flattening” the object so that it can be matched to a 2D image. Another way of thinking about it is that you have to tell Lightwave what point on the image corresponds to each corner of each polygon. This process is called *UV mapping*.

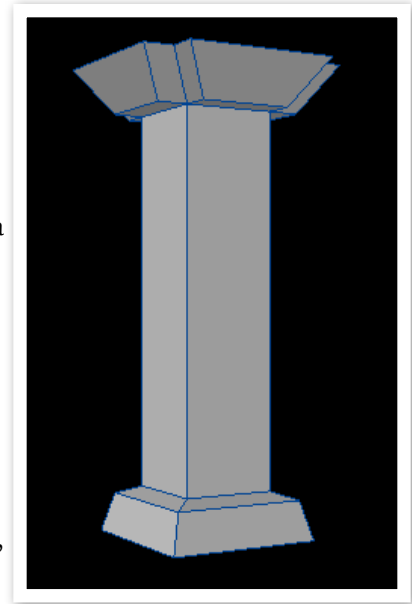
If that sounds laborious and painstaking, it is. But it yields spectacular results compared to the simplistic “planar” and “cylindrical” mappings mentioned earlier. And once you master this technique, there is no object you can't texture.

A model to practice UV mapping with

Before we can learn how to do UV mapping, we need an object to texture. We're going to use the column object described in *Introduction to Low-Polygon 3D Modeling for Games* from the Shockwave 3D Developer's Guide, located on the internet at <http://homepage.mac.com/nephilim/sw3ddev>. It is basically just a square column with variable radius, plus some pieces that jut out at the top, shown at right.

Deciding How to Texture

Before we do anything, let's think a little bit about how we want to apply textures to this model. There are symmetries here, so one thing to decide is whether or not the symmetries in the model are also symmetries of texture – are all the sides going to look the same, or will there be differences between each side?



There's a trade-off here to make. Making separate textures for each side will yield more realism and visual interest, but it will also take up more texture space. The more total texture area you have, the less texture resolution you can have. In other words, if you make all four sides are the same, then you can spend the texture space for the other three column sides on something else (or make the one side more detailed).

When considering things like this, take into account how the object will be viewed. Note that it will be impossible to see more than two sides of the column at once. Unless there is something very distinctive about the column sides, the player is probably not going to look at the south side of the column, and then go look at the north side of the column to see if it is the same. There is probably no compelling reason to make all four sides different, but there is value to making every other side slightly different.

So let's make the left and right sides of the column look the same, and the front and back sides of the column look the same. No one will ever see two identical sides of a column at the same time, and that should give enough variety to satisfy our needs.

Another consideration is how large to make the texture map. This is highly dependent on the game you are building. You should *never* make the texture much larger than it will appear on screen – that's just wasted texture space. In other words, there's no point in making a 512-pixel tall image for a column that will never be more than 100 pixels tall on screen. (This is why it's good to have a design document in place for your game before you start producing assets – if you don't know the answer to this question, then you don't know what texture size to use.)

Let's assume that the column will always be somewhere in the 200-250 pixel high range. So let's choose a texture size of 256 pixels tall.

Choosing a texture size

Why 256 pixels tall, and not, say, 200 or 250? Textures should always be created at “powers of 2” on a side. In other words, valid sizes for width and height are: 2, 4, 8, 16, 32, 64, 128, 256, and 512. Your images do not have to be square, but they do need to measure one of these numbers on each side. Thus, 16 by 256, 512 by 512, and 128 by 32 are all valid texture sizes.

(Now, technically, you don't *have* to create them at this size. But the engine is going to stretch them to the nearest larger size anyway. This means that you are losing detail *and* running the risk of the texture looking blurry or stretched in the final rendering. Luckily, the technique I am showing here, where we use Flash to create cartoony textures for objects, allows you to create your textures in a resolution-independent format, and the UV mapping process doesn't care about the actual size of the texture, so you can quickly swap out bad texture sizes late in a project if you have to, but it's still easier if you plan ahead and not have to do that.)

What about sizes larger than 512? You should consider 512 pixels the cap. While some video cards can handle sizes larger than 512, there are many which have 512 pixels as the highest resolution size. Your game won't break if you use higher pixel sizes, but textures will get blurry on computers with those lower-end video cards. (This is most often seen when people try to create overlays for their game with textures at 1024 by 1024 to cover an 800 x 600 screen, and then wonder why their heads-up display gets all blurry on some machines.) It's better to just design with 512 pixels as the maximum texture dimension, and avoid problems from the start.

The other consideration for texture size is that it is good if all the texture for a model fits on the same image. This is because we cannot use imagery from two image files without creating different surfaces, and thus, different “meshes.” The more meshes you have in a scene, the longer it takes to render in Shockwave3D, so it's a good idea to minimize the mesh count wherever you can for best performance. So in this case, it's a better idea to have both column side textures and the textures for the capstone and base in one file rather than one texture for the front and back sides, another for the left and right sides, another for the base, and another for the capstone. Since the object is tall and skinny, but will need some extra texture for the capstone and base, we should be able to get away with a 256 by 256 image to cover all the pieces.

Unwrapping the Model

Before we can create a texture, we need to unwrap the model. If you think of your model as a paper model, you can think of unwrapping the model as unfolding the paper to make it flat again. Let's take a look at how this is done.

UV Maps

To do this, we need to create what is called a *UV Map*. A *UV Map* is simply an assignment of (u,v) coordinate pairs to every vertex in your model. If you remember your high school algebra, (u,v) coordinate pairs are a lot like the (x,y) coordinate pairs you would use to graph lines and such. The u coordinate refers to a horizontal position on the image, and the v coordinate refers to a vertical

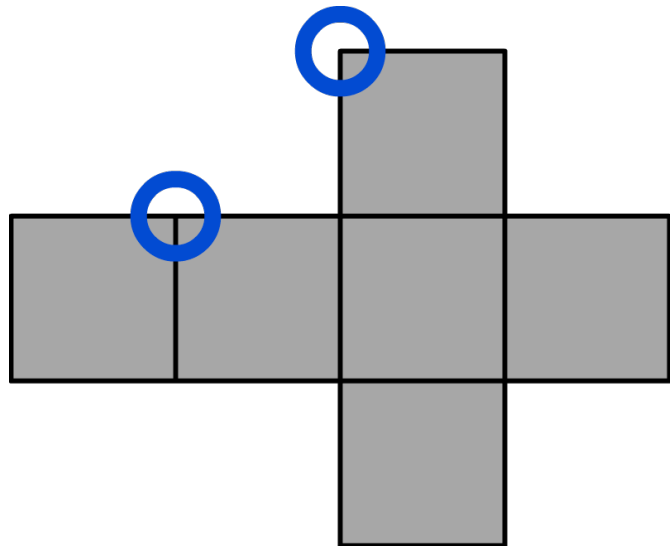
position on the image. Taken together, a (u,v) coordinate specifies a particular point on the image. In other words, it's "mapping" an image onto the 3D geometry.

Vertices and the UV Map

Before we create a UV Map, we need to talk a bit about how UV coordinates get assigned to vertices. If we just create a UV Map without any preparation, by default, each vertex in a model will be assigned to exactly one point on the image map.

The problem with this is that a vertex in a model often needs to be assigned to different locations on an image, depending on which polygon that vertex is used for.

To visualize this, imagine an unwrapped cube, shown at right. The two circled points on the image represent the same vertex, but two different places on the image texture. In order for us to properly unwrap and texture that cube, we would need to "unweld" that point, making three separate vertices, one for each polygon.



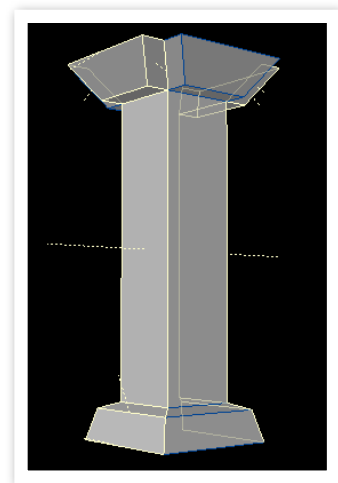
Once we have the textures assigned, we can merge the points back together, and Lightwave will remember the texture coordinates assigned to each polygon. Still, to make the best textures, we want our map to be connected, like the figure above. We want to only unweld the points at which we would cut the paper model, while the remaining points stick together into a solid piece.

Chopping up your Model in Preparation for your UV Map

Because of this, it's useful to think ahead a bit before embarking on UV Mapping. Let's think about how our pieces are going to be organized.

If you recall, we decided that our column will have the same texture on the left and right sides, and on the front and back sides. So, we'll need to texture each separately. Because the model is symmetrical, we can do some of the texturing once, and then just repeat it. Or, we could delete the geometry that is going to be duplicated, texture the symmetrical half, and then duplicate it – the texture information will be duplicated as well. Let's look at both methods as we go.

First, let's work on the left and right sides of the column. In perspective view, select all the polygons on the left and the right side of the column. Use the top view to ensure that you are selecting the sides instead of the front and back. If you followed along with the modeling tutorial, you should have 14 polygons selected; seven on each side.

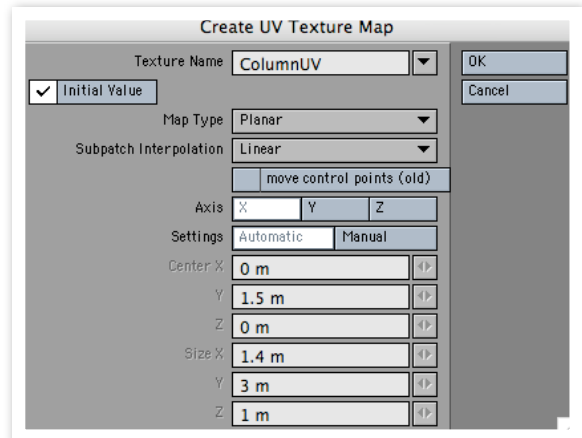


Cut and paste these polygons back into the object. What this does is chop your model into multiple pieces. When you cut the selected polygons, it breaks the vertices between your selected polygons and the unselected ones, and when you paste the polygons back in, it creates new vertices and doesn't try to weld them together. You've essentially "sliced" out the part you had selected.

We can now create a texture map for the left and right sides.

Creating a UV Map

So let's create a UV Map. Lightwave has a feature that does some of the mapping for us when we create the UV Map, so let's take advantage of that. Reselect the left and right sides (you can do this quickly by selecting one polygon on each side and pressing "]" to select all connected polygons), and then select "New UV Map" from the "Map" toolset. A dialog box will come up that asks you for some specifics of the new map, shown at left.



Enter "ColumnUV" as the texture name, select the "Planar" map type, and click the "X" axis. Leave the other elements at their default values.

The "Map Type" is the manner in which the vertices are given their initial (u,v) coordinates. The "Planar" type essentially creates the (u,v) coordinates as if you were looking at the model along the axis you select (in this case, the "X" axis). We select the "X" axis because we selected the left and right sides, which face along the "X" axis.

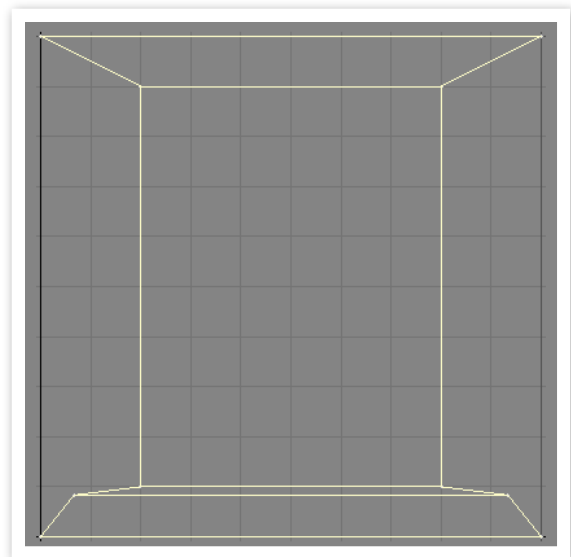
Press "OK" and your UV Map is created.

How Do I See My UV Map?

Now that you've created your UV Map, let's take a look at it. Let's change the upper left viewport to show us what our UV Map looks like. In the perspective drop-down menu at the top of the viewport, select "UV Texture."

The view will change from the view into the 3D world into a view of a 2D grid with a flattened version of your column displayed on it. If it's too small or large, press "a" to auto-zoom. It should look something like what is shown at right.

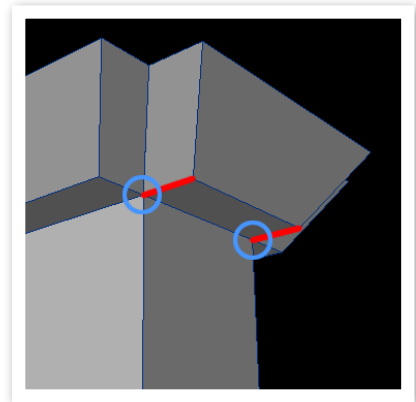
What you see here are the polygons that correspond to the geometry you selected, stretched to fill a square (which will eventually be your texture).



Try selecting the different polygons in this mode to see how they relate to the final figure. Note that you have *two* sets of geometry here – one for each side of the column. That’s because you had both sides selected, and both sides look the same when viewed from the X axis.

While exploring, you’ll probably notice that the shape of the polygons in the UV map is pretty different than the shape on the model. In fact, the downward- and sideways-facing polygons of the capstone are reduced to mere lines. That will make these polygons impossible to texture, so we need to go in and do some adjustments to the layout.

The first problem is that we have an unwelding we need to do: at the corner of the capstone. Notice how the sides of the capstone form a box, like the box described above? In order to flatten that capstone out, we need to “cut” along the bottom left and right edges of the capstone piece (along the red lines) so that the bottom part can be attached to the column side. To do that, we need to unweld the points shown circled in blue.

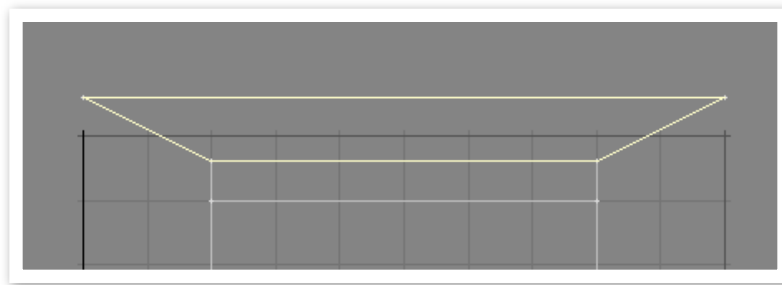


So let’s do that. Select these points, and select “Unweld” from the “Detail” tool set to unweld them (or type control-U). Make sure you do it to both sides. Now these points are unattached, and you can drag the geometry around separately.

The first thing we want to do is reveal the downward-facing polygon so it may be textured, so select the topmost three polygons of the capstone and drag them upward (using the “move” command: “t”) in the UV view.

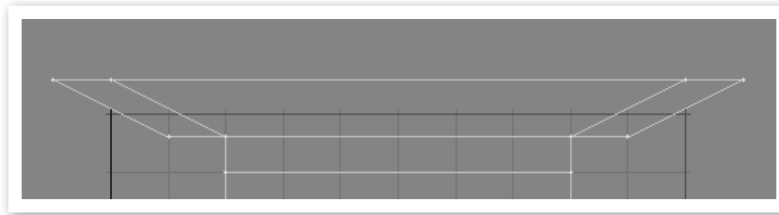
Note: *When you move, rotate, stretch, and resize things in a UV view, you are modifying the map, not the geometry. Be careful not to try to move things around in the 3D views, because that will mess up your geometry!*

When you’re done, you should have something that looks like this:



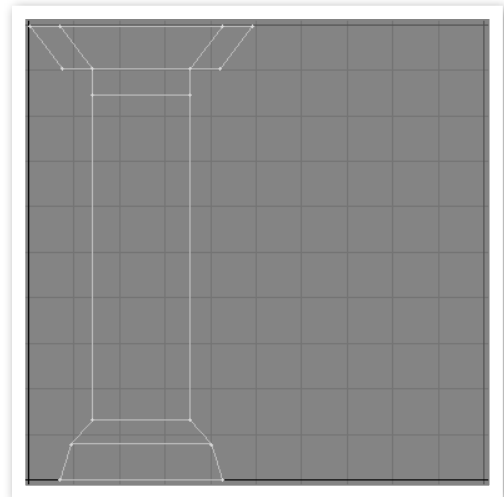
Now, we need to “fold out” the side flaps. Select the top two points on the lefthand side of the figure in UV mode, and then go over to the perspective view or front/back view and unselect the points at the fronts of the capstones. We now have selected the points on the side flaps of the capstones. Drag them to the left. Do the same thing on the right, and you will have pulled away the flaps for each side.

Your map should now look something like this:



Now, you can see that we have places where we can draw textures for the different polygons of the column side.

Now that we have it flattened, we just need to resize it. Use the “h” command and go down to the lower lefthand corner of the UV grid (so that 0%,0% shows up in the info box), and resize the figure to fit inside the box, and use the various point manipulation tools (“Move,” “Rotate,” “Stretch,” and “Size”) to tweak the column mapping to be as close to the proportions of the original model as possible. You should end up with something that looks like this:



There you have it: the left and right sides of the column flattened out and ready to be textured. Both sides use the same points on the texture, so they will be textured graphics from the same place on the image.

Taking Advantage of Symmetry

Now, we *could* go through this whole process again to assign (u,v) coordinates to the other pair of sides, but instead, why not take advantage of the fact that this model is symmetrical? Instead, let’s delete the geometry for the front and back sides, duplicate the (already textured) left and right sides, and rotate them into place. That will save us a lot of work.

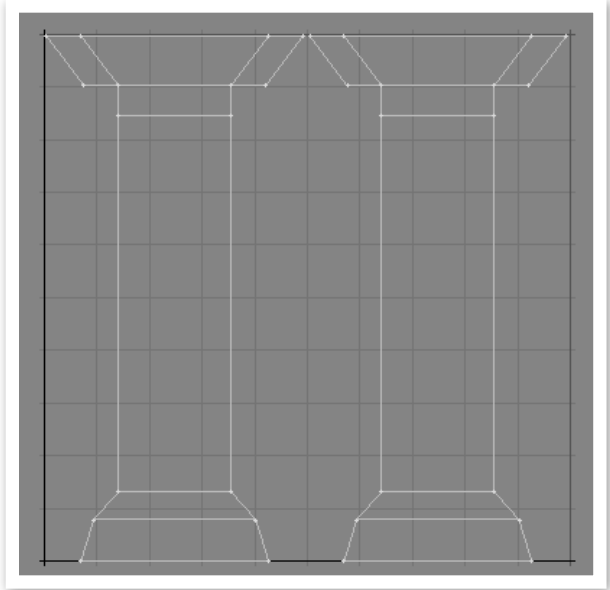
So select the front and back side polygons and delete them, leaving the left and right sides. Now select them and “Copy” them, but don’t paste yet. Since LightWave doesn’t automatically select what we paste, it’s easier to copy, then manipulate the original, and then paste a copy back where the original was.

But first, we need to move the (u,v) coordinates of the original. In the UV viewport, use the “Move” tool (“t”) to move the (u,v) coordinates to the righthand side of the grid, to leave room for the other side’s (u,v) coordinates when it gets pasted.


Select the “Rotate” tool (“y”), and in the “top” view viewport, move your mouse until the lower left information readout indicates your mouse is at (0,0). Then drag until it says 90 degrees. You have just perfectly rotated your model 90 degrees, and can now paste the side geometry from your clipboard. If you did it right, the column will be restored, but now you have all four sides having (u,v) coordinates.

Now, all that’s left is to merge the points of the geometry back together. Select “Merge Points” from the “Detail” toolset (or press “m”) and hit return. All the unwelded points that are sitting atop one another are merged into single points.

Now that you have flattened your model and assigned (u,v) coordinates to a 2D plane for its points, you can create a texture to apply to the surfaces.



Creating a Texture

The first thing we need is a reference guide for drawing our texture. The easiest thing to do here is to just maximize the UV view window (press the button that looks like this: , press “a” to auto-fit the grid, and do a screen capture of the entire grid (not just the polygons, but the entire grid, shown above as the black square - you may need to use your magnifying glass and move tools to get the entire grid on the screen). On the Mac, this is easily done by pressing Command-Control-Shift-4, which will let you drag out a rectangle to copy to your clipboard. This will give you a reference for drawing your texture. (Press the maximize button again to restore your previous view.)

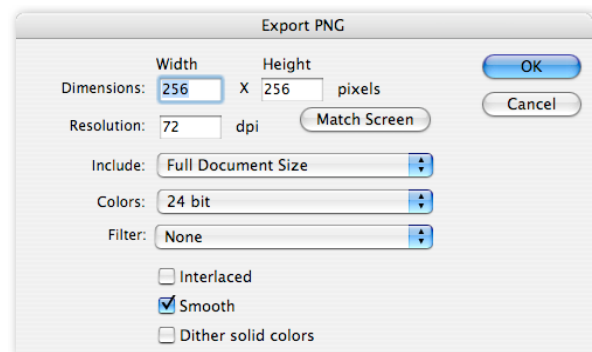
For this tutorial, we’ll create our texture in Flash, but you can use any image editing application you want. Create a new square document, 256 pixels wide by 256 pixels tall, and paste the screengrab into it so that it perfectly matches up with the edges. This will serve as your reference image. (In Flash, you can set the size and position exactly by using the Inspector. If you need help with this, there are many tutorials on the web and in books – using Flash to create graphics is beyond the scope of this document.)



Now, draw your textures over the template. Since you are just trying to set up your texture mapping, don’t spend too much time on it; just get something you can work with for now; once you get the mapping all set, you can really work on your texture and reload the texture in LightWave to get a quick preview of how it will look once assembled – it’s better to catch problems now than after you’ve done a lot of texture drawing.

When you’re done, knock all the background elements to black (or some other color appropriate for your texture if the texture bleeds). We chose black because we want cartoony black outlines along the edges of the column, but you could of course do whatever you want.

Now, you need to export the image for use in LightWave. The key here is to make sure it’s a power of two on a side (ours is 256x256), and that it doesn’t have an alpha channel. (Alpha channels cause problems with Shockwave3D – you can use them, but they’re tricky, and when you’re first getting started, it’s best to avoid them.) Shown at right are the image export settings from Flash (we’re exporting a PNG) – other image editor applications will probably have similar options.

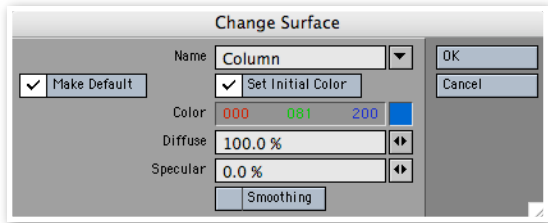


Applying the Texture to the Object

We need to create a “surface” that the texture gets applied to. A “surface” is just a texture definition that collects all the texture information and gives it a name.

In LightWave, make sure there are no polygons selected (so that all polygons are selected), and click the “Surface” button at the bottom of the screen (or press

“q”) to bring up the Surface dialog, shown at right. Let’s call our new surface “Column,” make sure the Diffuse and Specular values are at 100% and 0% respectively (for a matte finish), and click “OK”.



You will see your object turn whatever color was selected. You have created a surface called “Column” and applied it to all the polygons you had selected. Now we need to apply our new texture using the texture map to it.

Call up the Surface Editor from the upper left-hand navigation pane (or press F5).

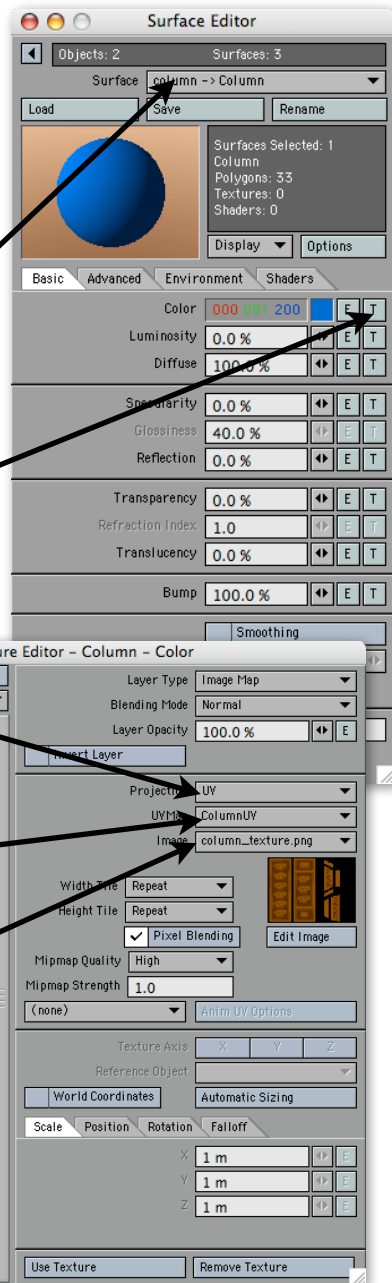
First, make sure you are working with the right texture by selecting “Column” from the “Surface” popup at the top of the window.

Now we need to apply our drawing as a texture. Since we are defining the “color” property of the texture, we will apply the drawing as a texture of the color property. Click the little “T” button to the far right of the “Color” property.

This brings up the Texture Editor window. Most of the controls in this window may be ignored. For now, select “UV” from the Projection drop-down.

The window will rearrange itself a bit to present options for UV projected textures. Select the UV Map you created (called “ColumnUV” if you have been following the names suggested) in the UVMap dropdown menu (this just tells LightWave which UV Map to use, since you can have several).

Then select “(Load image)” from the Image dropdown, and locate the column texture file you saved from before. Confirm, and you will see your texture image appear in the Texture Editor window, and it will show up on the column object, as well!



Use the perspective view rotation tool to rotate your object and check for any poorly-textured places. If everything has come out correctly, you're done! Otherwise, you can go in and tweak the UV map, or go back and change your texture image.

Properties of Materials and the Mathematics of Light

We've just seen how to apply images to an object to give it variable values for a property. Specifically, the "color" property. But there are several other properties of an object that you can manipulate to create different visual effects. Let's talk a little bit about how to define the various properties of an object to get the effect we want.

Bear with me here while we talk a bit about how the color of an object gets computed. This involves a little math, but luckily, the computer is there to do it all for you – all you need to know is how to put textures together to achieve the desired results.

When the computer tries to figure out what color to draw a pixel when rendering a 3D world, it starts out by considering what object is being seen at that pixel. This pixel has a several qualities associated with it: the material of the surface, the direction the surface is facing, and the lights that are pointing at it.

Once these are determined, it sets about applying rules to determine the final color value to present. It helps to understand how these rules work to achieve the effect you want. Here are some general rules of behavior to keep in mind:

- When determining what lights are hitting a surface, the color values of the lights are added. Thus, a red light and a green light pointed at a point on an object is the same as a yellow light pointed there.
- When determining the resultant surface color to display, the surface will only reflect up to the amount of light of that color. For instance, if the surface is blue, and you point a green light at it, it will look black, because there is no blue in the green light to reflect. Or, if you point a white light at a black object, it will be black. (Discounting specularly and emissiveness.) For this reason, it is almost always a good idea to use desaturated colors for objects. Desaturated colors have at least some red, blue, and green in them, and so, will always react to lights pointed at them – you can make a light blue without losing all the visibility of your brown walls.
- The angle between the light and the surface acts as a divisor; polygons facing a light directly will get all the light, while polygons facing perpendicular to the light (or further) will get none of the light.
- Director's surfaces respond separately to ambient light (i.e., the light available in the shadows), normal light, and specular highlights. Each can have a different color value, so that an object



can reflect only the reds out of the shadows, only the blues out of the light, and only the greens from highlights. (Not that you'd want to do that.)

- Emissive textures are “light-emitting” properties. They are added on top of whatever lighting the other properties give it, as if the surface glows at that point. This is good for, say, running lights on starships, or skyboxes. Note that emissive textures don't actually throw off light; they will not illuminate any other objects in the scene; you will have to add actual lights for that.

How Lightwave and Shockwave3D describe the properties of materials

Both Lightwave and Shockwave3D define a series of material properties that describe how the material reacts to light. Unfortunately, the characteristics that they use are different; there is a close mapping, but it is not perfect. In particular, because Lightwave is also used for high-end video graphics, it supports many more texturing capabilities than Shockwave3D does, so you need to be aware of what things Shockwave3D cannot do before spending time on texturing.

All of these properties are available for change either in LightWave (using the surface editor panel shown earlier, where you can set uniform values or add a UV-mapped image for variable values) or from within Director itself (using scripting to change the values at run-time).

Light Wave	Director	Notes
Color	Diffuse	This defines the base color of the object. If you use a texture, the object's color is taken directly from the texture; the color you select is irrelevant.
Diffuse	(none)	The Diffuse property of Lightwave does not appear to have an effect when exported. Instead, Shockwave3D takes Lightwave's Color as the diffuse value.
Luminosity	Emissive	This defines how much the material “glows.” While this does not mean the object throws off light, it does let you, for instance, allow it to be seen even where there are no lights pointing at the object (and thus, would normally be completely black). In Lightwave, this value is a percentage, while in Shockwave3D, it is a color. The value in Shockwave3D for the color when exported from Lightwave is the Color property multiplied by the Luminosity . Textures assigned to Lightwave's Luminosity are ignored upon export, but you can programmatically assign a texture-based emissive value with advanced Lingo.

LightWave	Director	Notes
Specularity	Specular	This determines how bright the highlight is on the material. Again, in Lightwave, this is a percentage value, and in Shockwave3D, it is a color. When exported, the specular value is white scaled by Lightwave's Specularity .
Glossiness	Shininess	This determines how large the specular highlight is. Smooth objects (high values) have small highlights, while matte objects (low values) have large highlights. If you use a texture, it defines what parts of the texture are reflective.
Reflection	(special)	The Reflection property does not apply in Shockwave3D, but if you apply a texture, it becomes an environment map which, in combination with Glossiness , determines what the reflection on an object looks like.
(none)	Ambient	The Ambient property of Shockwave3D defines how the object looks in shadow. It has no exportable equivalent in Lightwave.
Smoothing	(none)	This defines how the vector normals are calculated; they can either look smooth or faceted. This property is reflected in Shockwave3D, but is not controllable as a property directly.

Any properties not listed for Lightwave3D are not exportable. In particular, none of the texturing options in the other texture tabs (“Advanced,” “Environment,” and “Shaders”) are exportable.

Closing Thoughts

While the preceding information should get you started, there is no substitute for tinkering. The more you work with 3D textures, the more you will come to intuitively predict how things will look within the Shockwave3D engine. But you can only get there by experimenting.

In particular, you should try to hone your skills in “baking” 3D information into your models’ textures. The more detail you can achieve with textures, the less geometric detail you need to build (and the less geometry Shockwave3D will have to compute each frame).

For example, you might try adding creeping vines climbing up your column. If you were to model those out, it would take a lot of extra geometry, but if you can effectively “bake” them into the texture, then the model will have exactly the same rendering time, but will have more evocative detail.

Remember – time spent on texturing that doesn’t improve the game is time wasted, so only obsess over your texture mapping when you need to make sure seams line up perfectly or when the objects are going to be closely inspected by the player. There are many techniques that allow your textures a little “slop” without affecting the final product much (especially with cartoony textures – those black outlines can hide a lot).