

# Texture Effects

*All too often textures are overlooked as a solution for creating effects. The tendency is to think of them merely as a means for coloring objects. The tutorials in this chapter demonstrate some ways in which engaging visual effects can be created quickly and easily simply by taking advantage of the power that textures offer. The most versatile of all texture nodes—the Ramp texture—will be put to work in many of these examples.*

# 1

---

## Chapter Contents

Creating Animated Lighting Effects with Ramp Textures  
Streaking Energy Effects  
Layering with Multiple UV Sets: The Zombie's Hand  
Abusing Ambient Occlusion: The Sci Fi Scanner  
Controlling Particles with Textures: Swimming Bacteria

## Creating Animated Lighting Effects with Ramp Textures

🌀 In this first scenario, our art director has provided us with a scene consisting of a UFO hovering above a tree-lined hill in the wilderness. Our task is to add some brilliant flashing lights to the craft. Open the scene `saucer_v01.mb` in the Chapter 1 folder on the CD and take a look at what's going on.

You can see that the saucer model has been roughly animated over a hill. We have a nighttime sky and some scraggly paint effects trees (see Figure 1.1). It looks like a pretty simple scene. The main direction we have been given is to add lights that circle the perimeter of the model as well as something interesting for the dome on top. Yes, directions from some art directors can actually be this vague, depending on the project. We could start by modeling lights from simple spheres and attaching glowing shaders to each one and then use “set driven keys” for them to circle around, but the art director has just instructed us that she needs to post this for client review by lunch time. So let's find a quicker way to do this. Rather than add a bunch of little lights, let's make part of the model one big light and use an animated Ramp texture to create the circling glow.

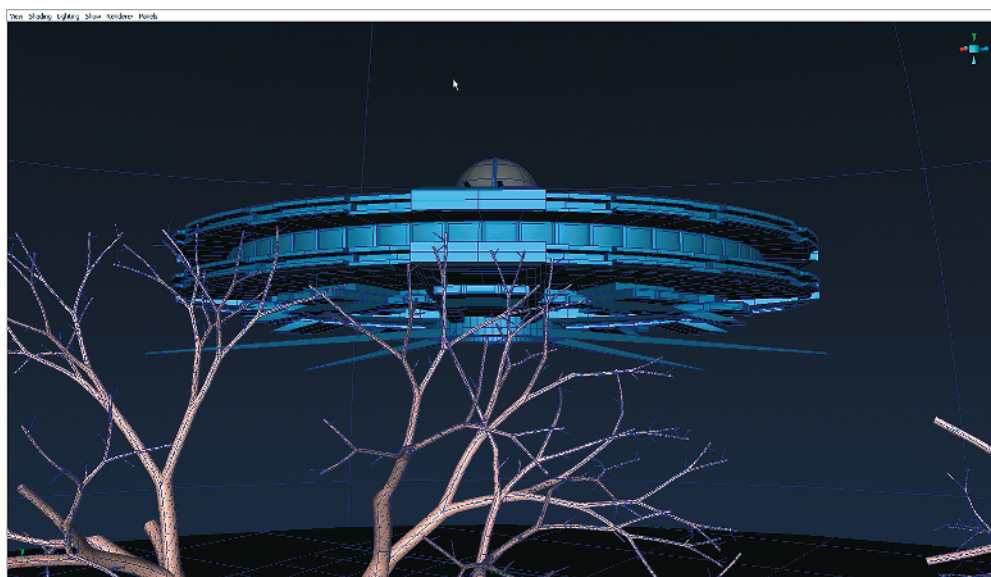


Figure 1.1: The saucer model hovering over the paint effects trees

To create the glowing effect, we will be using the surface shader. As far as shader types go, the surface shader is just about as simple as you can get. It's basically a shader that does not react to lights or shadows in the scene; it just applies a flat color to an object.

This is why they can be ideal for creating lights on an object as long as those lights are pretty simple. The shader does not emit light in the scene when the scene is rendered with standard Maya software. When a color is placed in the **Out Glow Color** channel, the resulting effect is a very brilliant glow. Its simplicity is why I prefer it for creating the kinds of effects described in this chapter. You can do the same kind of thing with a Lambert or Blinn or any of the other shaders by plugging a color into the **Incandescence** channel; however, this often gives you a few more attributes to keep track of when tuning the glow. It really comes down to how much you want to deal with and the particular effect you are trying to create.

## Creating the Light Geometry

First we need to create some geometry which will become our light source.

1. In the Display Layer Editor, turn off the visibility for all of the layers except the **UFO\_layer** to make the scene easier to work with.
2. Rewind the animation to the beginning where the UFO is level with the grid.
3. Take a look at the model. The most obvious place for the lights is in the midsection. It looks like the modeler intended us to use the square polygons that circle the midsection of the model (Figure 1.2).

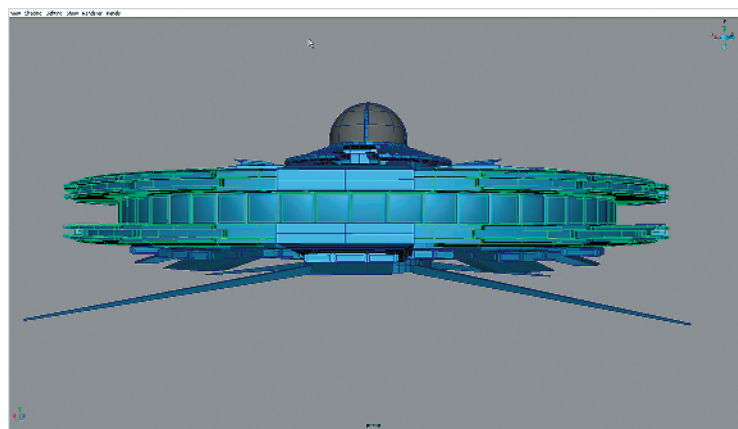


Figure 1.2:  
Zooming in on the  
model shows some  
square polygons around  
the perimeter; these will  
become our perimeter.

4. In the Outliner, expand the **UFO** group, find the **saucer** polygon object, and select it.
5. Use the **Paint Selection** tool to select each of the square polygons that circle the saucer. In Maya 8, the **Paint Selection** tool has been added to the standard toolbar just below the **Lasso** tool. When you activate the tool, the saucer section will switch to component mode. In the status line, click the **Select Face Component** button to restrict the selection to polygon faces.
6. When you have them all selected (as in Figure 1.3), double-check that you didn't select anything extraneous; switching to wire frame (hot key = 4) is sometimes a good way to do this.



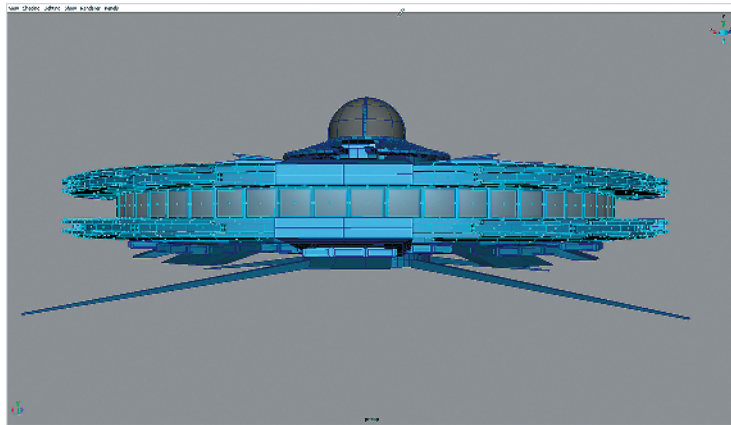


Figure 1.3:  
The square polygons  
around the perimeter of  
the saucer are selected.

7. We could apply a shader directly to these faces but let's duplicate them and scale them up a little just so we don't have to mess with the main UFO model's UVs. To do this, switch to the Polygon menu set and choose **Edit Mesh > Duplicate Faces > Options**. In the options, make sure **Separate Duplicated Faces** is checked.
8. Click the Apply button. With the object still selected, open the Channel Box (shown in Figure 1.4), select the **polyChipOff1** input connection (it may already be selected), and scroll down to find the **Keep Faces Together** attribute. Make sure it is set to on. Find the **Local TranslateZ** channel about midway up in the Channel Box and set it to 0.1. This will move the duplicated polygons out a little from the surface of the UFO.

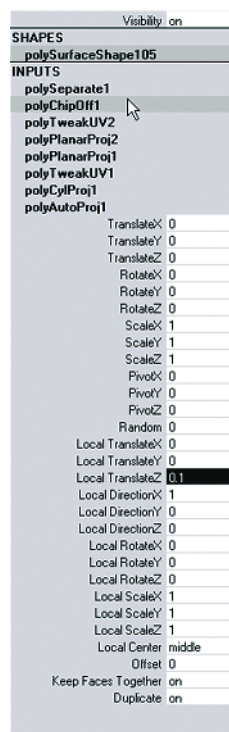


Figure 1.4:  
The Channel Box settings for  
the **polyChipOff1** node

9. Switch back to the object selection mode (hot key = F8). In the Outliner you'll see that the saucer object is now a group (Figure 1.5). In the group there is a list of poly objects labeled polySurface1, polySurface2, and so on. If you select polySurface1, you'll see that it is the saucer object. PolySurface2 through polySurface59 are the duplicated polygon squares. Select polySurface2 through polySurface59 in the Outliner and use **Mesh > Combine** to make them one object.

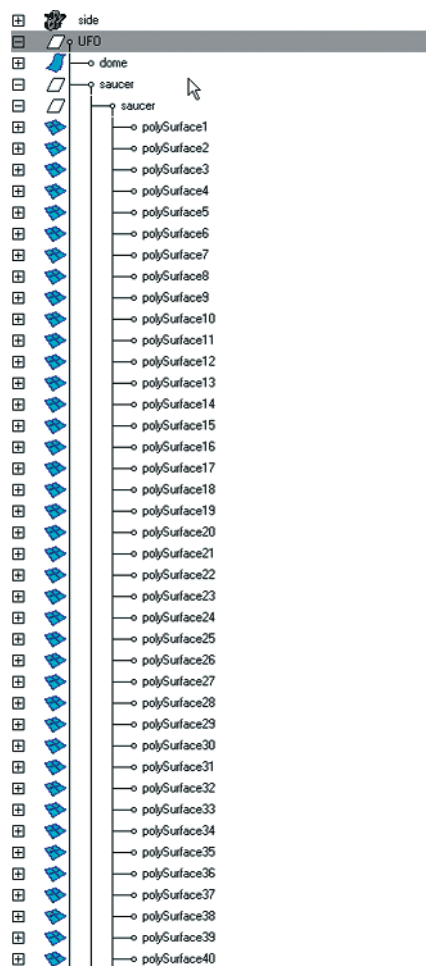


Figure 1.5:  
The Outliner showing the  
new duplicated faces

10. Select the saucer (which has been renamed polySurface1) and the new polySurface60 (at the bottom of the Outliner) and delete history on these objects. The empty transform nodes created when you combined the polygons should disappear from the Outliner.
11. Rename polySurface1 “saucerHull” and polySurface60 “hullLights.”
12. Move the hullLights object back into the saucer group.

## Shading the Lights

Now we're ready to make some lights.

1. Select the `hullLights` object and choose **Windows > UV Texture Editor** to open the UV Texture Editor. Press the F key to frame all of the UVs.
2. Select the UVs and choose **Polygons > Normalize UVs**. The UVs stretch to fit the positive coordinate section of the UV grid. The edges may be a bit wavy (see Figure 1.6); you can straighten them out if you'd like, but it really doesn't matter for this purpose.

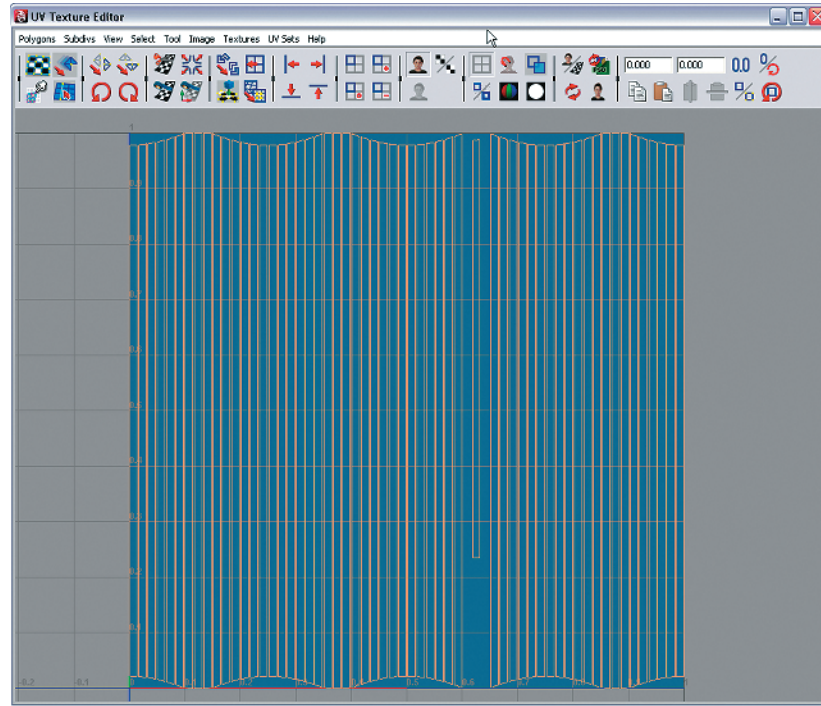


Figure 1.6: The normalized UVs for the `hullLights` object

3. Back in the perspective window, select the `hullLights` object again and assign a surface shader to it. Name the surface shader "`hullLightsSG`."
4. Open the Attribute Editor for the surface shader. In the **Out Color** channel, click the checker button, and in the **Create Render Node** panel, click the **Ramp** button to assign a ramp. Name the ramp "`hullLightsRamp`."
5. Press the 6 key to switch to hardware texturing in the perspective view. You should see that our UFO has a nice festive array of rainbow-colored lights. We need the color band to go around the saucer hull; to fix that, open the Attribute Editor for the ramp and change it from a **V ramp** to a **U ramp** (Figure 1.7).

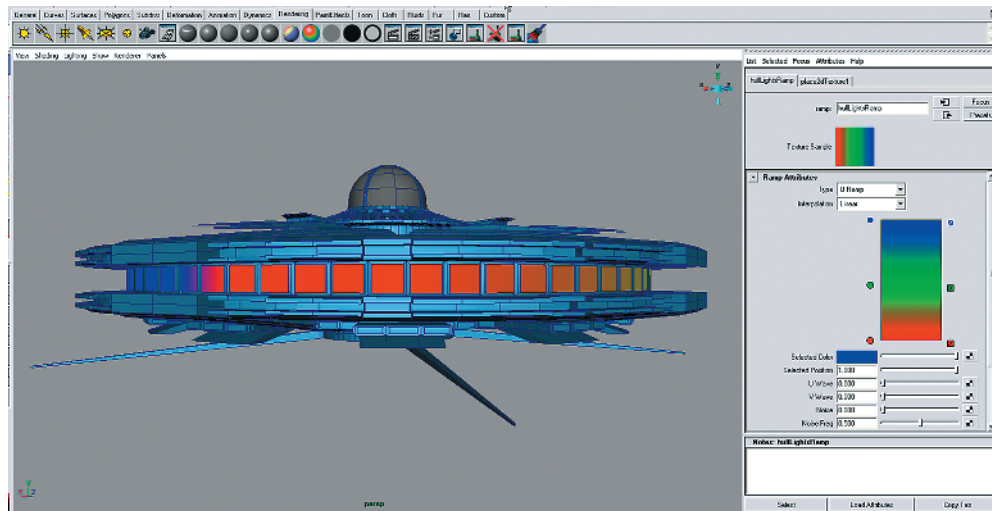


Figure 1.7: Switching to a U ramp makes the colors wrap around the perimeter lights in the correct direction.

6. Much better. Now edit the ramp so that the top is yellowish orange and the bottom is black. Delete the green color in the middle by clicking on the box next to the green area on the ramp display in the Attribute Editor. Move the black color marker halfway up the ramp.
7. Tumble around the model and see how the lights fade from bright orange to black. Animating the lights around the ship is extremely easy.
8. In the Attribute Editor, switch to the place2D Texture node for the ramp texture. Rewind to the start of the animation, right-click over the first field in the Translate Frame attributes (this is the field for the U coordinates' position), and choose **Set Key**.
9. Move to frame 30, change the 0 in this field to a 1, and set another key.
10. Hit the arrow next to the Translate Frame attributes in the Attribute Editor. In the **Post Infinity** menu for the **Anim Curve Attributes**, choose **Cycle**. In each of the boxes under **inTan Type** and **outTan Type**, change the tangent type to linear by typing the word *linear*.
11. Play the animation; you'll see the ramp circle around the ship (Figure 1.8).

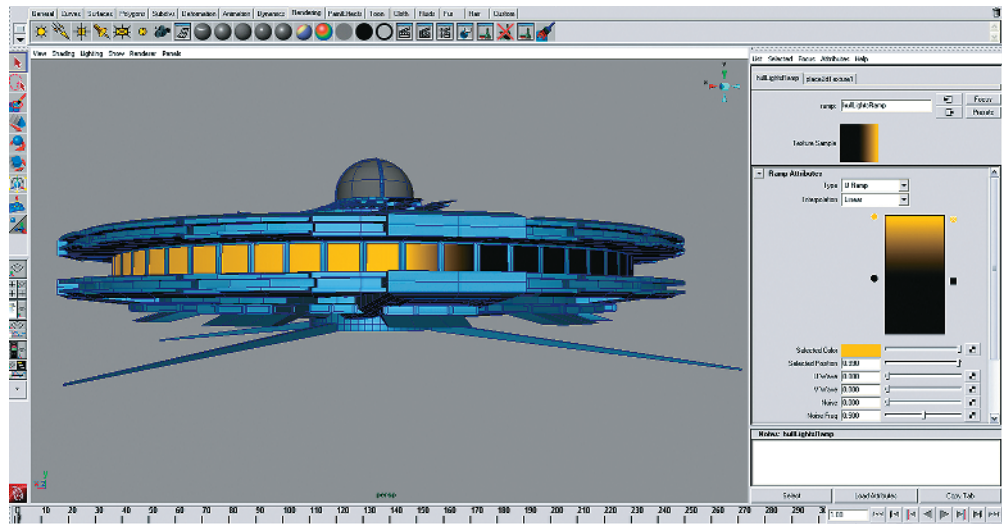


Figure 1.8: The lights now circle around the perimeter of the saucer.

## Creating Glowing Lights

Now let's pump these lights up a bit. They look good, but it's kind of Close Encounters of the Blah Kind.

1. Open up the Hypershade and select the `hullLightsSG` node. Right-click over the swatch and choose **Graph Network**.
2. Select the green line between the `hullLightsRamp` and the `hullLightsSG` surface shader. Hit the Backspace key to delete it.
3. MMB drag the ramp over the surface shader, and choose "Other" to open the connection editor.
4. In the Connection Editor, select **Out Color** on the list on the left and **Out Glow Color** on the list on the right. In the perspective window, the lights will turn black. That's OK; they will glow when you render them.
5. Now turn the visibility of the display layers for the background objects back on and render the scene. There are some eerily glowing lights for you! (See Figure 1.9.)

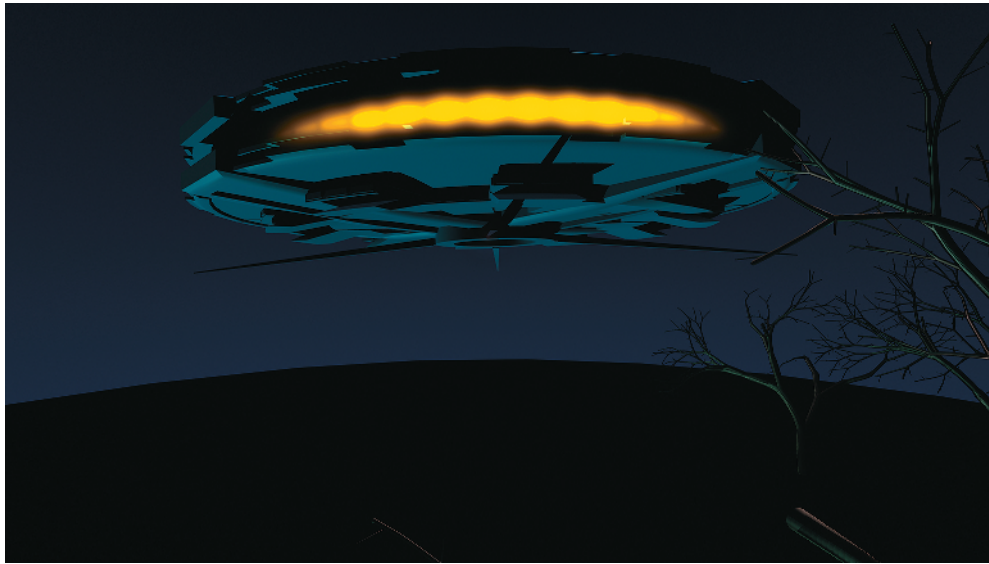



Figure 1.9: The lights now glow like the lights on a UFO should.

When using shader glow in a scene, you may encounter a flickering problem when the sequence renders. This can be fixed by opening up the shader glow node (found in the Hypershade) and turning off the **Auto Exposure** attribute. However, this will make your glowing effects much stronger and blown out. To fix this, render a still with the **Auto Exposure** option in the shader glow node turned on and then open up the output window and make a note of the **Glow Intensity Normalization Factor** value and the **Halo Intensity Normalization Factor** value. Copy these values into a text editor. Then turn off the **Auto Exposure** option on the shader glow node and copy the **Glow Intensity Normalization Factor** value into the **Glow Intensity** field and the Halo Intensity Normalization Factor into the **Halo Intensity** field. The resulting render should match your original render, but if you render an animated sequence, the flickering will be removed. It is important to remember that the shader glow node in the Hypershade sets the overall glow attributes for all incandescent shaders in the scene. Each scene can have only one shader glow node. You can adjust the glows on individual objects by changing the **Glow Intensity** or the **Incandescence** color on their shaders.


 After rendering out a quick preview (check out saucer1.mov on the CD), you can see that the lights are looking pretty good. Of course, now the art director pops up behind us and says, “Wow, that’s cool, can you add more lights on the bottom? And make the animation of these new lights a little more interesting.”

No problem! We can use the same technique but this time will shake it up with some clever use of UV mapping.

### Alternative Light Displays

Creating variations on the light displays is easy. In this section we’ll see how changing the UV mapping on the light geometry changes the way the lights behave.



1.  Open saucer\_v02.mb from the Chapter 1 folder on the CD. Here we are with the same scene and our perimeter lights are circling around. If you'll notice, on the bottom of the saucer there is a new row of duplicated polygons constructed in the same way as the perimeter lights. In the Outliner, this object is a child of the bottomStructure group. It is named "bottomLights."
  2. Select the bottomLights object and open the UV Texture Editor. You'll see that the UVs have been laid out in a horizontal row.
  3. Just as with the perimeter lights, assign a surface shader to the bottomLights object and drop a ramp in its **Out Color** channel. Name the surface shader "bottomLightsSG" and the ramp "bottomLightsRamp." Set the ramp type to **U Ramp**.
  4. Adjust the ramp so that it's bright red at the top and fades to black about halfway down.
  5. Place some keyframes on the ramp's **Translate Frame** attributes for its place2DTexture node just as you did with the perimeter lights. Remember to set **Post Infinity** to "Cycle". Set the **In Tan Type** and **Out Tan Type** to Linear.
- Now the lights move around this bottom section. It is slightly more interesting since the lights follow a path that conforms to the extruded shape of the UFO's bottom section.
6. Select the bottomLights object and open the UV texture editor.
  7. Starting on the left-hand side, select the UVs of every other polygon face and move them upward out of the row, but make sure they are pretty much in line (Figure 1.10).

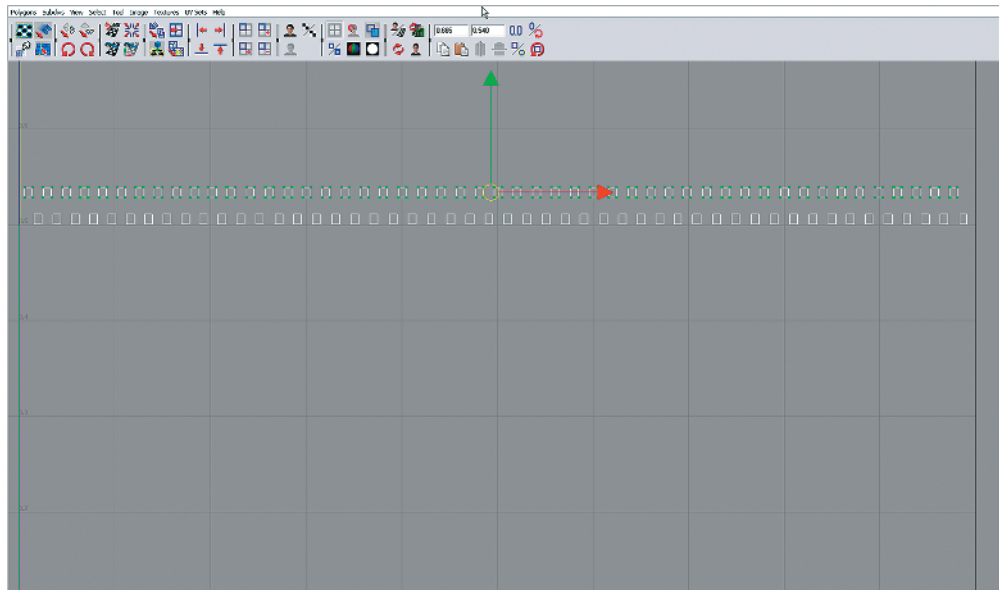


Figure 1.10: The UVs for alternate faces have been selected and moved up in the UV Texture Editor.

8. Once you have two distinct rows of UV faces, select the top row and use the Scale tool to invert the UVs sideways (alternatively, you can use the Rotate tool, just as long as this row of UVs is reversed from its original direction).

When you play the animation, you'll see the lights now alternate so that the sequence goes in two directions at once. You can use the animated ramp technique in conjunction with the UV layout to come up with endless variety of light patterns guaranteed to hypnotize even the most jaded UFO abductees. When you're happy with your light animation, disconnect the ramp from the out color of the surface shader and hook it up to the out glow color just as you did with the perimeter lights.

## Shading the UFO Dome

For a final touch to the UFO, we'll add a psychedelic light display for the saucer's dome using another ramp texture.

1. Select the NURBS dome on the top of the UFO. Assign another surface shader to this geometry. Name the new surface shader "domeLightSG."
2. Drop a ramp in the color channel of the dome. Name this ramp "domeRamp."
3. Make sure the ramp is set to **V Ramp**.
4. Adjust the ramp so that it has three bands of lime green alternating with black; make sure the top and bottom are black (Figure 1.11).

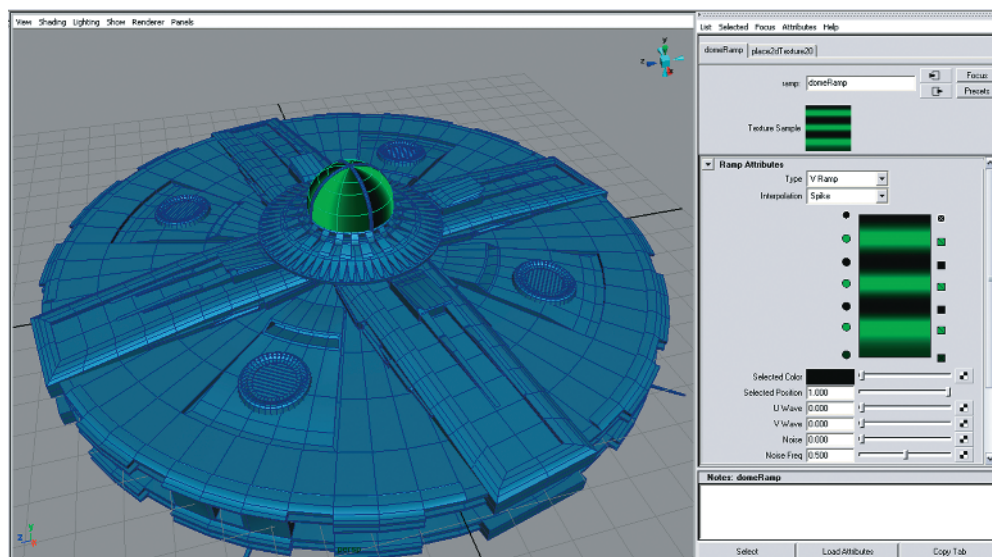


Figure 1.11: The ramp with three bands of green

5. Set **interpolation** to Spike so the bands are more defined.
6. Move to the place2DTexture node for this ramp and set the second field in the **Repeat UV** attribute to 3; this will increase the number of green bands on the dome.
7. Set the **Rotate Frame** value to 40 so that the bands become kind of a swirl.

8. At frame 0, right-click over the second field in the **Offset** attribute and set a key.
9. At frame 30, change the value to 1 and set another key.
10. Hit the arrow next to the offset value and in the **Anim Curve Attributes**, set the Anim Curve attribute's **Post Infinity** to **Cycle**, and set the **In Tan Type** and **Out Tan Type** to **Linear**.

Play the animation. You'll now see the dome swirling with green light—perhaps these aliens hail from some kind of '60s Go-Go planet.

For the bottomLights object and the dome shaders, you can plug the ramps into the **Out Glow** channel just as we did for the perimeter lights. To see a completed version of the scene, open *saucer\_v03.mb* from the Chapter 1 folder on the CD.

### Further Study

The exercises in this chapter really highlight the usefulness of the ramp shader and it just barely scratches the surface. The next tutorial takes these ideas to another level. Before moving on you may want to try creating alternative light display patterns by altering the UV layouts of the perimeter and bottomLights. The ramp texture applied to the dome offers up another area of experimentation; see how many variations you can create by playing with the attributes on its *place2DTexture* node. Try switching it to a **U Ramp** and see what other types of effects that leads to.

## Streaking Energy Effects

Fresh from the success of the flying saucer shot, the art director has returned with another job. In this next scenario, a cartoon robot is seen hurtling through space. The shot calls for the robot to leave a trail of streaking, glowing energy in its path. The idea of using particles to create this effect might leap to your mind; certainly that is a viable option, but before we go down the road of tweaking emitters, fields, and goals, let's see what we can get away with using our humble ramp texture.



### Know Your Glows

Glowing effects in Maya can be created in many different ways, from Optical Effects and Light Fog to the incandescence channel on a shader. It's a good idea to experiment with different ways of creating glowing effects in Maya. Find the methods that best suit you and develop them over time. Open up the *glowTypes.mb* scene in the Chapter 1 folder of the CD and take a look at the various settings. These are just a few of the ways glowing effects can be created in Maya.

The *glowTypes.mb* scene contains five different spheres with different types of glowing and incandescent effects applied. Each sphere has been placed on a different

display layer. Turn off all of the layers except the blinnWithIncandescence layer and do a quick render. In the render view, store the rendered image and then go through and render each layer individually. Store the images in the Render View window (in the render view menu, choose **File > Keep Image In Render View**) and compare the results using the scroll bar at the bottom of the page. Then turn the visibility of all the layers on and render the spheres together. Note how the glow from one shader can bleed to the next depending on the angle of the camera (Figure 1.12). Take a look at the settings on the shaders applied to each sphere.

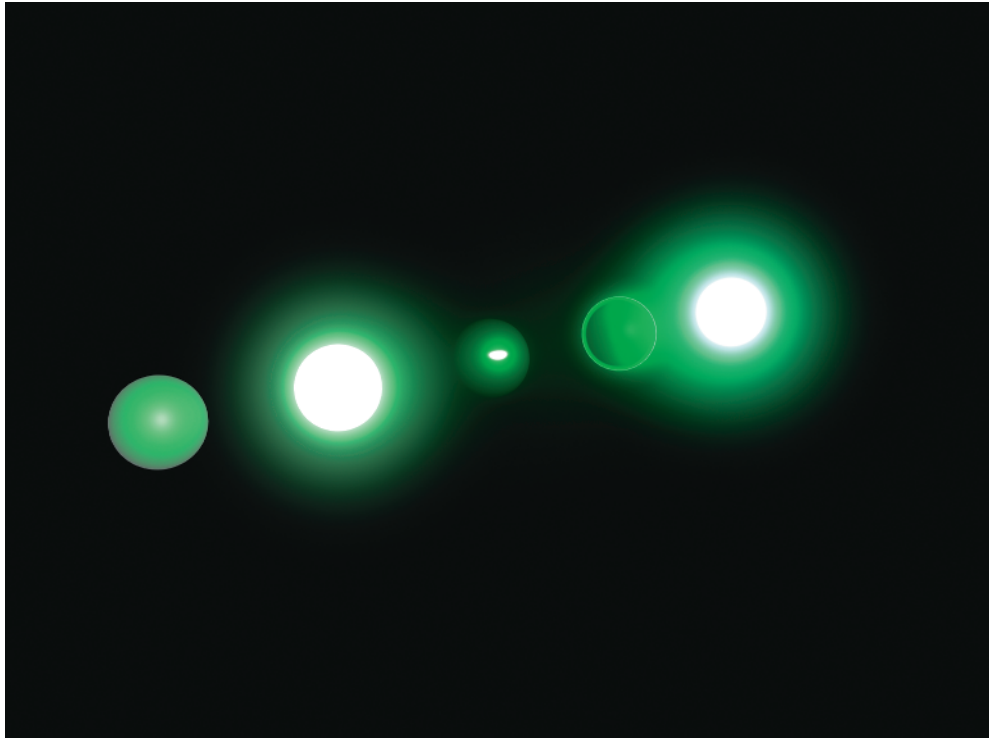



Figure 1.12: The glowTypes.mb scene shows five spheres with different types of glowing effects applied.

 Open the robot\_v01.mb scene from the Chapter 1 folder on the disc. Set your perspective window to camera1 and play the scene. What you'll see is a little manga-style robot flying along a motion path with the camera following. Set the Time Slider to frame 54 and do a test render. You'll see that he has a toon shader applied to his body and a glowing surface shader applied to the polygons that make up his eye. If you look in the Outliner, you'll see a pfxToon1 object. Select this object, turn its Visibility setting to On, and do another render.

Toon outlines have been applied to the object; this is represented by the pfxToon1 node. They look cool when rendered but they can slow the scene down in the OpenGL display. Once you've seen what the robot looks like with the toon lines, you can open the Attribute Editor for the pfxToonShape1 node and uncheck the **Display In Viewport** attribute, which will hide the toon lines in the perspective window but still allow it to render.

We can take advantage of the robot's motion path curve and use it to extrude a tube that will align itself to the flight path. The energy effect will be a texture applied to this tube. If we leave history on for this tube, then if changes are made to the robot's motion path, the tube will update automatically. Less work for us!

## Creating the Energy Geometry

To create the streaking energy we'll first create a tube along the flight path of the robot. This tube will have a glowing texture applied to it in a later step.

1. First let's create the tube. In the Outliner, select the nurbsCircle1 object, Shift+select the motionCurve object, and switch to the Surfaces menu set. Choose **Surfaces > Extrude**. In the options, make sure you check the buttons for **Tube**, **At Path**, **Component**, and **Profile Normal**. Make sure **Curve Range** is set to **Complete** and the **Output Geometry** is **NURBS**.
2. A NURBS tube now appears along the motion curve. Select this surface and apply a surface shader to its texture. Name the shader "energyTrailSG."
3. Open the Attribute Editor for this new surface shader and click the Apply Texture button next to the **Out Color** channel. From the texture window, choose a Ramp texture. The tube now turns into the familiar red, green, blue rainbow. What a happy flight our robot is having.
4. Name the ramp "energyRamp" and switch its type setting to **U Ramp** so the colors flow along the tube.
5. Adjust the ramp so that it has three thin lines of orange and yellow separated by black (Figure 1.13).

These colored lines will be our energy streaks, but so far they look pretty goofy. They span the entire length of the tube and the robot doesn't even fit in the tube very well. What we want is for the colored streaks to trail behind the robot. We don't want them to appear in front of him.

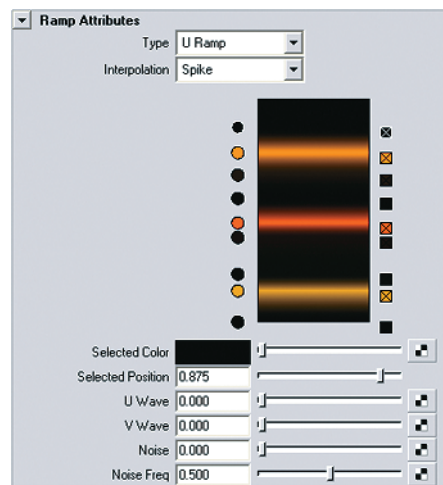


Figure 1.13:  
The ramp for the robot's  
streaking trail of energy

## Creating a Mask with a Second Ramp

To hide the parts of the tube that appear in front of the flying robot we'll create a second ramp and use it to mask the colors of the first ramp.

1. Open up the Hypershade, select the energyTrailSG surface shader that has been applied to the tube, and choose **Graph > Input And Output Connections** from the Hypershade menu to graph the network.
2. Break the connection between the ramp and the shader's **Out Color** channel (select the green line between the nodes and hit the Delete key).
3. Create a new ramp in the work area and connect it to the shader's **Out Color** channel. Name the new ramp "maskRamp."

We are now back to our happy rainbow tube with the colors appearing in bands along the tube. Open up the Attribute Editor for the maskRamp texture.

Notice that the ramp is a V ramp. That's great—just what we want. We changed the last ramp to a U ramp so that the colors would streak along the tube. This ramp is going to control where those streaks appear on the tube; it will in fact act as a mask and be layered on top of the previous ramp.

4. Delete the blue color at the top of the ramp. Change the green to black and the red to white.
5. Set the **Interpolation** of the ramp to None. If you look in the perspective window, you'll see that half of the tube is white and the other half is black. Select the black color marker on the ramp and move it up and down; notice how the black section of the tube expands and contracts when you do this.

In order to use this ramp as a mask, we want the white area to determine where the streaking energy will appear and the black area to determine where it will be masked. The area in which the streaks appear will then have to grow along the tube at the same rate as the robot flies along the path, so what we need to do next is animate the black part of the tube so that it diminishes in size as the robot flies along the path. We could select the black color marker on the ramp and set keyframes on its **Selected Position** attribute on the ramp, but that is kind of crude. Instead, we'll make a connection between the robot's position and the black marker's position on the ramp. This is amazingly easy.

## Connecting the Ramp to the Motion Path Output Value

We'll make a connection between the robot's position on the motion path and the position of the color marker on the maskRamp so that the masking of the energy color is automated.

1. First let's take a look at something interesting: As you move the black marker up and down the ramp, the value in the **Selected Position** field change. When it is toward the bottom of the ramp, the **Selected Position** value is closer to 0; when it is toward the top, it is closer to 1 (Figure 1.14).



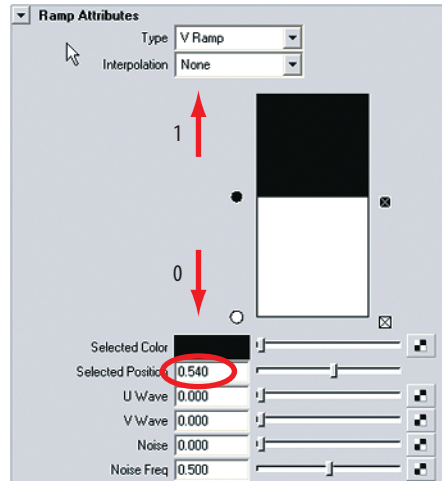


Figure 1.14:  
Changing the value in the  
ramp's Color Position

2. In the Outliner's Display menu, deactivate the check mark next to **DAG Objects Only**. The Outliner now shows all the nodes in the scene. Scroll down the Outliner and select the motionPath1 node. The motionPath1 node was created automatically when the animator who set up this scene attached the robot to the curve with an animated motion path. Open the Graph Editor with this node selected and select the **U Value** attribute. Take a look at the graph. (You may want to hit the F key to set the focus of the graph on this attribute; it will make it easier to see what's going on.) At the start of the animation, the **U Value** is closer to 0 when the robot is at the start of the path. At the end of the animation, the **U Value** is closer to 1 when the robot is toward the end of the path. The motion path works by animating an object along the U coordinates of a NURBS curve. We can take advantage of this by making a direct connection between the motion path's **U value** attribute and the position of the black marker on the ramp. To do this, we'll use the Connection Editor.
3. Open the Hypershade and graph the input/output connections for the energy-TrailSG surface shader on the tube.
4. From the Outliner, drag the motionPath1 node onto the work area of the Hypershade (Figure 1.15).

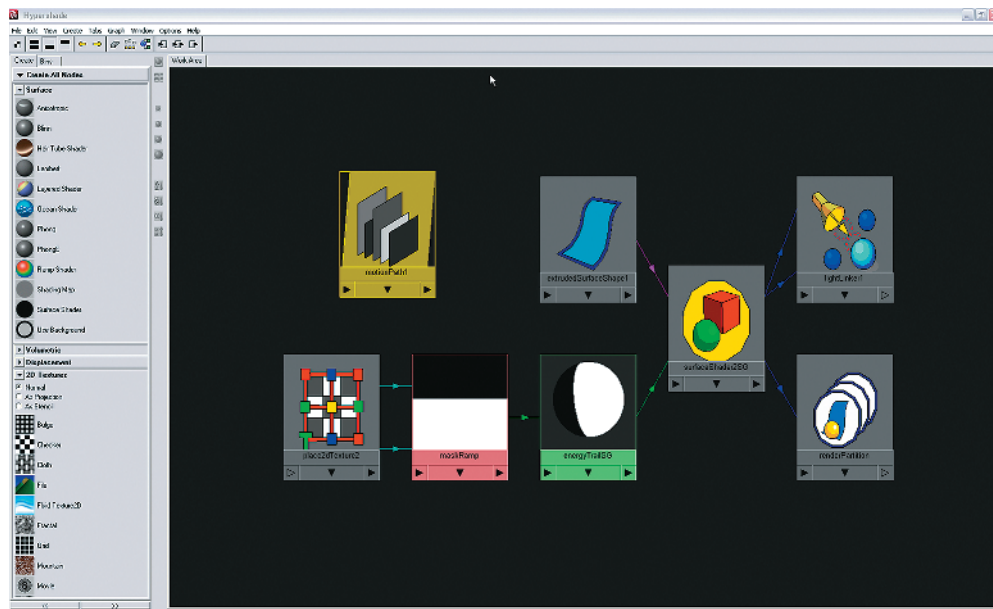


Figure 1.15: The motionPath1 node in the work area of the Hypershade.

5. From the menu at the top of the Hypershade, choose **Graph > Rearrange Graph** so everything becomes clearer.
6. In the Hypershade work area, MMB drag the motionPath1 swatch over the maskRamp swatch.
7. From the pop-up menu, choose **Other**.
8. The Connection Editor will open with motionPath1 on the left and maskRamp on the right (Figure 1.16).

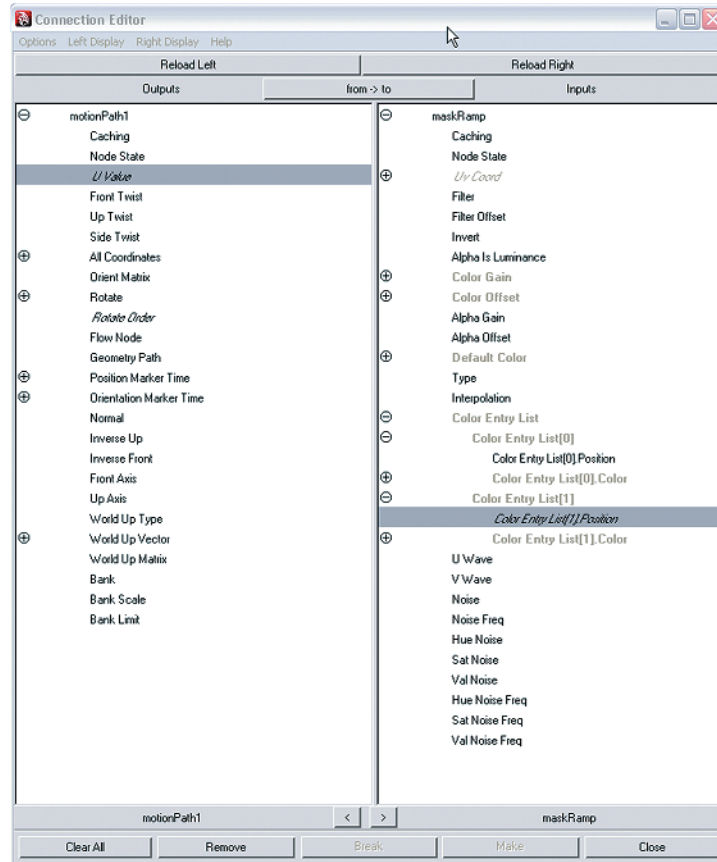


Figure 1.16: You can use the Connection Editor to hook up the motionPath1's **U Value** attribute to the position of the color marker on the maskRamp.

9. Click on the **U Value** attribute under the motionPath1 attribute list.
10. Under the maskRamp attribute list, expand the attribute group labeled **Color Entry List**. This corresponds to the color markers on the ramp. The bottom marker is usually labeled 0, and then additional markers are labeled 1, 2, 3, and so on. Expand the attribute list under **Color Entry List[1]** and select **Color Entry List[1].Position**. This will connect the **U Value** of the motion path to the position of the black marker on the ramp. Close the Connection Editor and take a look in the perspective window (Figure 1.17).

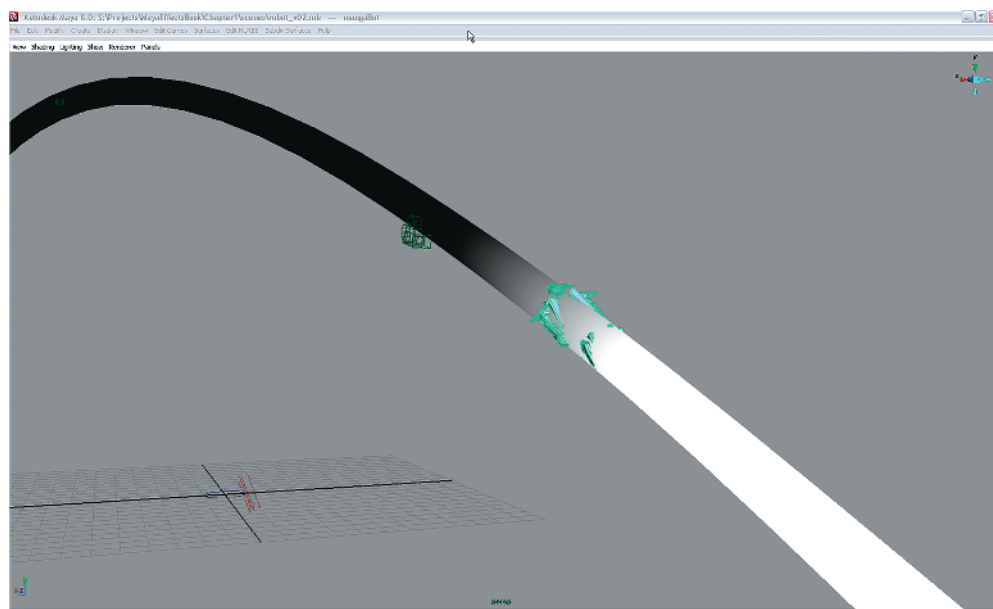


Figure 1.17: The colors on the tube now correspond to the robot's position.

As you move the Time Slider back and forth, you should see the white part of the tube move along with the robot. Making the connection between the two means less work if the timing of the robot needs to be changed later on. The white part of the ramp will always follow the flight of the robot. However, you may notice that at the very first frame of the animation, the entire tube may be white. What happened? At that point of the animation, the robot is at the very beginning of the path, the **U Value** of the motion path is 0, and thus the position of the black marker on the ramp is also 0 and so is the position of the white marker on the ramp. Maya decides to override the black color with the white color when both values are 0. The best way to fix this is to get the **U Value** of the motion path to be just a little above 0 at the start of the animation so that the black and white color positions are not on top of each other on the ramp.

11. Select the motionPath1 node from the Outliner and open the Graph Editor. Select the **U Value** attribute from the bottom of the list on the left.
12. Select the first keyframe on the graph, and in the **Stats** fields at the top of the Graph Editor, change the value in the right field to .001. This will set the keyframe above a value of 0. The tube should turn black. Play the animation and make sure the white part of the tube follows the robot. It may look fairly “steppy” or jerky as it moves, but the rendering will be smooth; it’s just the OpenGL display that appears steppy.

## Reconnecting the Energy Shader

Now that we have our mask working correctly, we need to reconnect everything in the energy shader.

1. Open up the Hypershade and from the materials area, drag the energyTrailSG surface shader applied to the tube down to the work area using the middle mouse button.
2. Switch to the Textures tab and drag the energyRamp down to the work area.
3. Select the energyRamp, Shift+select the energyTrailSG shader, and graph the input/output connections of the two nodes.
4. MMB drag the energyRamp over the energyTrailSG surface shader and choose **Default** from the pop-up menu. This will automatically disconnect the maskRamp from the **Out Color** channel and replace it with the energyRamp.
5. Open the Attribute Editor for the energyRamp. Take a look at the **Color Gain** attribute in the **Color Balance** rollout. By default, the **Color Gain** is white, meaning the colors for this texture are at full strength. If we lower this value to gray, the colors will dim—kind of like a brightness control for the colors that make up the texture. If we lower this value to black, the colors will disappear completely. We'll use this property to make our mask.
6. From the Hypershade, middle-mouse-button-drag the maskRamp over the **Color Gain** attribute in the Attribute Editor. This will place this texture in the **Color Gain** channel (Figure 1.18).

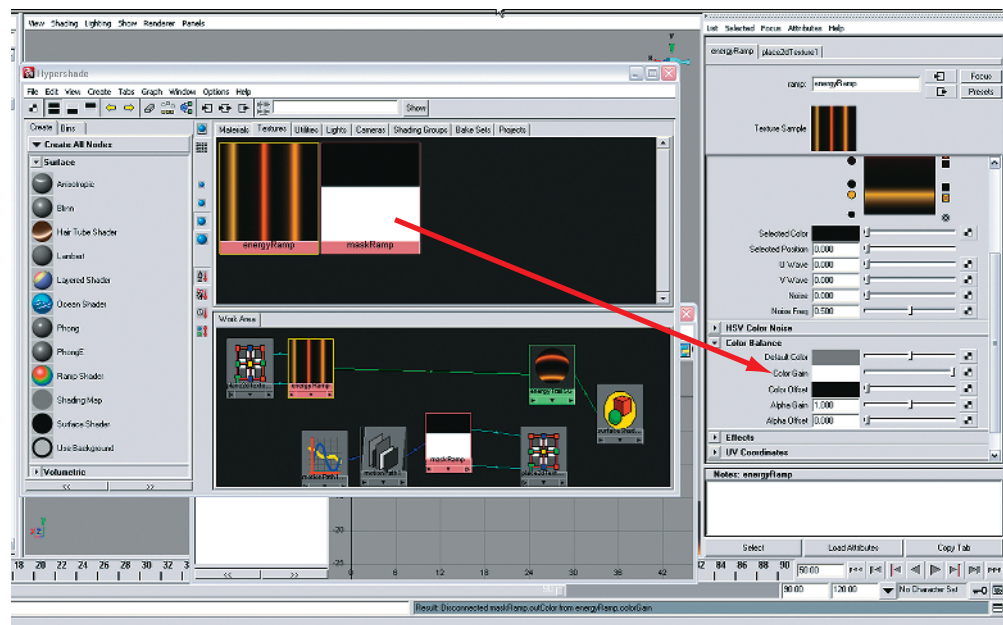


Figure 1.18: Middle-mouse-button-drag maskRamp from the Hypershade on to the color swatch for energyRamp's color gain in the Attribute Editor

Take a look at the perspective window; you can now see that the streaks follow along behind the robot.


## Making It Glow

The last thing we have to do is change the texture so that the streaks look more like glowing bands of energy.

1. In the Hypershade, graph the input/output connections for the energyTrailSG surface shader applied to the tube. Select the green line that connects **energyRamp.outColor** to **energyTrailSG.outColor** (hold the pointer over the lines connecting the swatches in the Hypershade to see the labels). Hit the Delete key to break the connection.
2. MMB drag the energyRamp texture over the surface shader. From the pop-up menu, choose **Other**.
3. Select **Out Color** from the list on the left side and **Out Glow Color** from the list on the right.
4. Close the Connection Editor and open up the Attribute Editor for the surface shader. Set the **Out Transparency** color to white.
5. From the Hypershade, switch to the Materials tab and select the shaderGlow1 node. Open its Attribute Editor.
6. Turn off **Auto Exposure**, set the Time Slider to 35, and do a test render.

Wow, that robot is really on fire! Let's tone down the effect a little using the attributes on the shader glow node. Try these settings (listed are the settings I adjusted; you can leave the others at their defaults for now):

<b>Glow Type:</b>	Exponential
<b>Halo Type:</b>	Exponential
<b>Threshold:</b>	0
<b>Glow Intensity:</b>	0.18
<b>Glow Spread:</b>	0.025
<b>Glow Eccentricity:</b>	0.085
<b>Halo Intensity:</b>	.033
<b>Halo Spread:</b>	.015
<b>Halo Eccentricity:</b>	0.463

 When you're happy with your settings, turn the visibility of the pfxToon lines back on and render the sequence or play the QuickTime of the animation located on the CD. The trail may look a little blocky. To smooth it out you can open the Attribute Editor for the extrudedSurface1 node and increase the settings under Simple Tessellation Options. Open robot\_v02.mb from the Chapter 1 files on the CD to see a finished version of the scene (Figure 1.19).



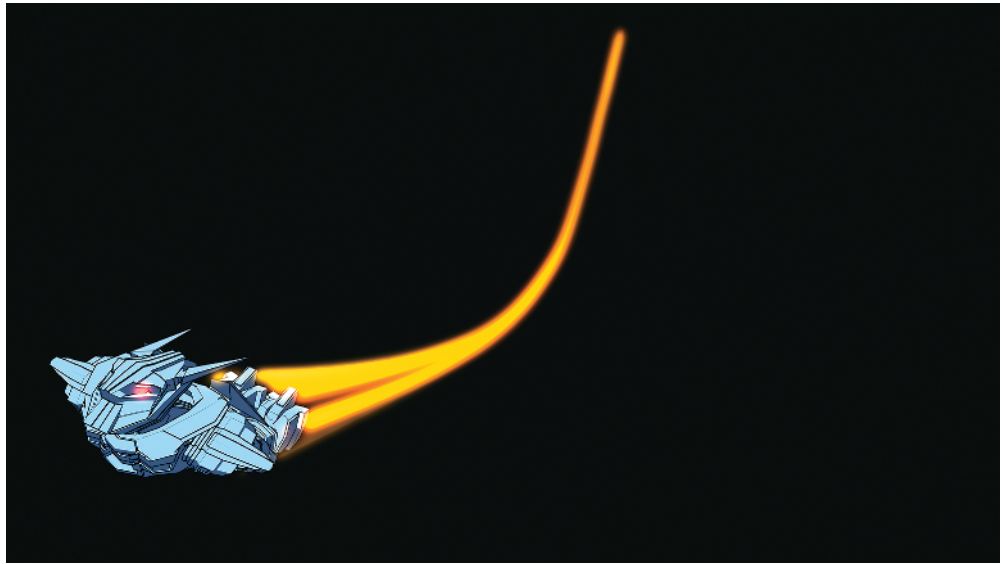


Figure 1.19: A render of the finished robot scene


### Further Study

You can use the **Color Gain** on **maskRamp** to tone down the glow if you still feel that it is too much. You can try setting the **U Wave** value on the **energyRamp** to 1.0 to add the appearance of the energy twisting around the tube. You can also either add more color bars to the **energyRamp** or raise its **repeat U value** on the **place2D** texture node above the value of 1 to get more bands of color in the energy.

## Layering with Multiple UV Sets: The Zombie's Hand

On to zombies! Our art director has a new shot involving a close-up on a zombie's hand. The zombie is clearly in the early stages of zombification. We need to add a disgusting black veiny creeping texture that will seal the deal on this shot. Our modeler and texture artists have provided us with a delightful hand model with creepy textures included. However, we will not be using any animated texture file sequences; instead, we are limited to the hand's color texture and a single frame of the black veiny infection texture.



1.  Open the `zombieHand_v01.mb` file from the Chapter 1 folder on the CD and check out the model. Select the hand model and open up the UV Texture Editor.
2. You can see the UVs laid out over the color texture that has been created in a 3D paint program. It doesn't look much like a hand. The main part of the hand is in the upper left and the five fingers are spread around on the right and bottom.

Right-click in the UV Texture panel and choose UVs from the marking menu to switch to the UV mode. Drag and select around each piece and note in the perspective window the corresponding pieces of the hand (Figure 1.20).

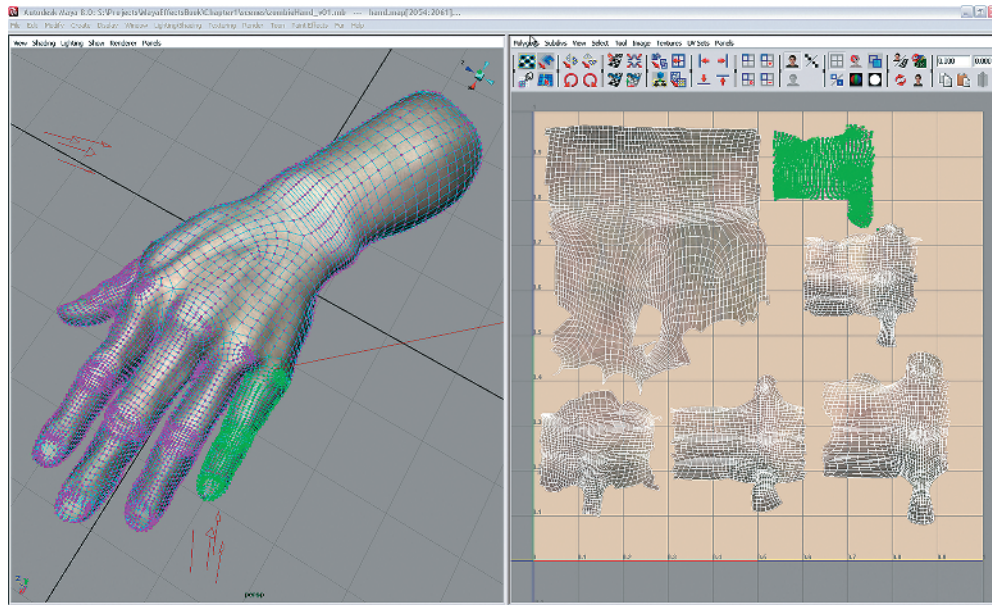


Figure 1.20: Selecting one area of the UVs causes one of the fingers to be highlighted in the perspective window.

This UV layout is OK for texturing but not great for creating our effect. The reason is that the area where we want the infection effect to be—the back of the wrist—is fairly small in this UV layout. So much of the texture coordinates have been devoted to the fingers. We don't want the infection to spread to the fingers in this shot, mainly because as the texture moves over the spaces between the wrist UVs and the finger UVs, the seams will be very obvious. Fortunately there is a way around this—UV sets.

## Creating Multiple UV Sets

A polygon model in Maya can have more than one UV set, which adds a great deal of flexibility for creating texture effects (as well as texturing in general). To achieve this effect, we will use a procedural texture to create a mask that will reveal the second color map (the veiny one), gradually creating the effect of a spreading infection. We will use a layered texture in the color channel of the shader applied to the hand and a Ramp texture to create the mask. To maximize the area where the Ramp texture will be applied we will first create a new UV set for the object. This is easy to do, but the Maya interface for UV sets is not the most elegant one around.

1. First let's create a new UV set based on the UV coordinates for the wrist. In the UV Texture Editor, select some of the UVs in the large group in the upper left, the wrist area. With a few selected, you can right-click over them, scroll down in the marking menu, and choose **Select > Select Shell**. All of the UV coordinates for the wrist area should now be highlighted in green.

2. You can copy these UVs into a new empty UV set. To do this, go to the top menu bar in the UV Texture Editor and choose **Polygons > Copy UVs To UV Set > Copy Into New UV Set > Options**. In the options, type the name “infection.”
3. All the UV texture coordinates for the fingers should disappear. They still exist; it’s just that now you are looking at a whole new UV set with just the coordinates you had selected and copied. Confusing, isn’t it? This is certainly an area of the interface that could be improved in future versions of Maya. In any case, just to make things clearer, turn off the texture display in the UV Texture Editor by clicking the face icon in the upper menu area.
4. You can switch back and forth between UV texture sets by choosing **Polygons > Set Current UV Set**. You have to type the name of the UV set into the field. The original set is called “map1,” and the new one is called “infection.” Try switching back and forth a few times. When you are comfortable with this workflow, switch back to the infection UV set.
5. Now we’ll normalize the UVs for the wrist so they fill up the upper-right portion of the UV Texture Editor. Marquee select all the UVs in the wrist area and choose **Polygons > Normalize UVs** to do this (Figure 1.21).

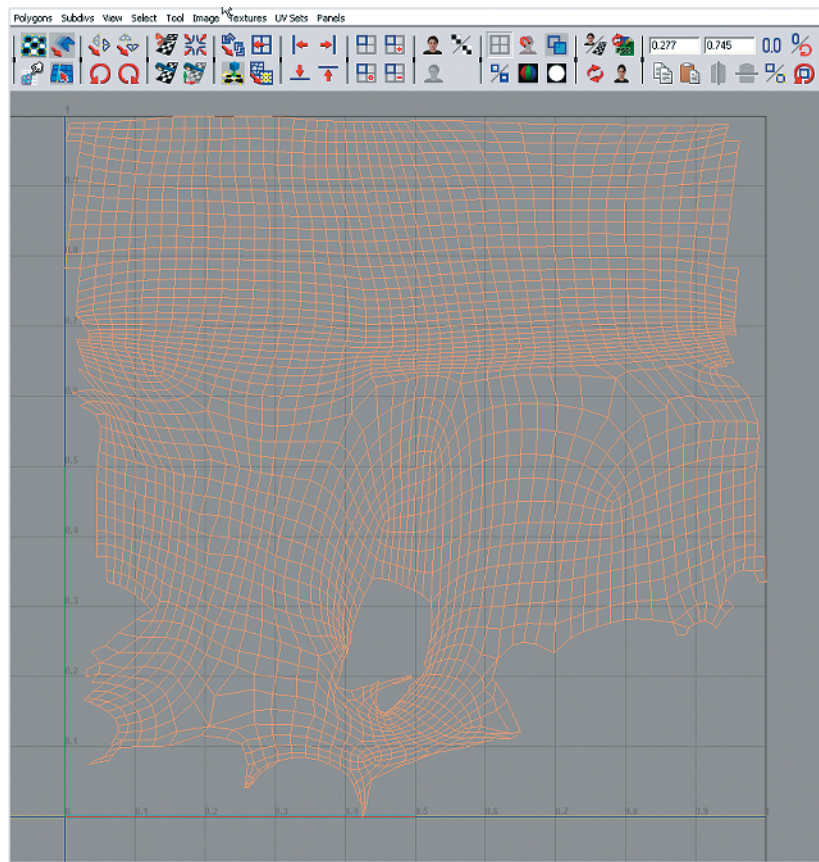


Figure 1.21: The UVs for the infection UV set have been normalized.

Notice that the texture on the hand has not changed in the perspective view. This is because we are editing the coordinates for the infection map but the color texture on the hand's shader is still linked to map1. This is exactly what we want. If the color on the hand has changed and things look really funky, it's most likely because you did not switch to the infection set and you are currently editing the map1 set. You may have to hit Undo a bunch of times or start over to fix this.

## Creating the Mask for the Spreading Infection

Now that we have our UV sets created (that wasn't too hard), let's create the spreading mask effect.

1. Open up the Hypershade, select the handSG shader, and graph it in the work area. It's a very simple network: just a color and a bump texture.
2. Create a Ramp texture in the work area and drag it over the handSG group. Choose **Color** from the pop-up menu. This will automatically disconnect the painted color texture from the handSG group. The hand now turned to a happy rainbow color (Figure 1.22). Zombification has been abated! He's cured!

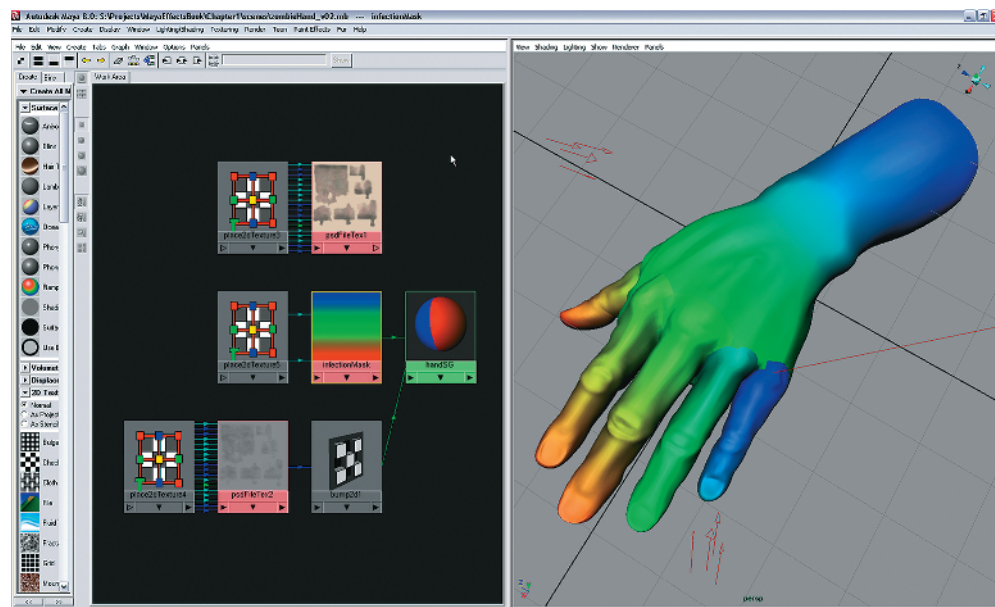


Figure 1.22: The Ramp texture has replaced the File texture applied to the handSG's color channel.

3. The rainbow color on the hand follows the map1 texture coordinates, so you'll notice the seams where coordinates for the fingers start. We'll switch the ramp so that it uses the infection UV coordinates. First change the ramp1 texture's name to "infectionMask" to make things clearer. You can open its Attribute Editor and enter the new name in the box at the top.
4. To assign the infection UV set to the infectionMask Ramp texture, select the hand model and open the Relationship Editor. To do this, choose **Window > Relationship Editors > UV Linking > Texture-Centric**.



5. In the Relationship Editor (Figure 1.23), you'll see the handSG group on the left with a list containing the infectionMask texture and psdFileTex2 texture (that's the texture assigned to the hand's bump channel; we disconnected psdFileTex1 from the color channel). On the right side of the panel you'll see the infection and map1 UV sets listed. If you click on infectionMask or psdFileTex2, you'll see map1 highlighted on the right. Click on infectionMask on the left and then infection on the right to change infectionMask's UV set. You'll see the rainbow colors shift on the hand model in the perspective window. Close the Relationship Editor when you have done this.

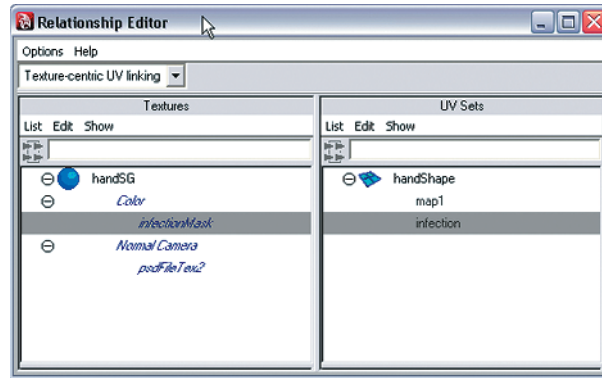


Figure 1.23:  
The Relationship Editor  
allows you to change the UV  
set for a given texture.

6. Open the Hypershade and graph the network applied to the hand. Notice that there is a new uvChooser node applied to the place2Dtexture node for the ramp. It gets created automatically when you have a shader with multiple UV sets. Click on the infectionMask ramp and open its Attribute Editor.
7. Delete the green color marker; change the red color to black and the blue color to white.
8. We're going to change this black and white gradient to a splotch in the following steps. First, move the white color marker halfway down the ramp.
9. Change Interpolation to None.
10. Change the type of ramp to **Circular**.

Now we have a big black dot that's covering half the hand. If you move the white color marker on the ramp up and down, you'll see the dot grow and shrink. It may be hard to see this because the dot wraps around the hand.

11. In the Hypershade, select the place2D texture node attached to the ramp and open its Attribute Editor. Set the **Coverage** to 0.5 in both U and V. The hand becomes mostly gray. Select the hand and take a look in the UV Texture Editor. Make sure that the Display Image button is turned on.
12. You'll see that the white square with the black dot—our ramp—has moved to the lower-left corner of the texture coordinates for the UVs. We want to move it closer to the upper right. Notice that it's gray outside the white box of the ramp (Figure 1.24).

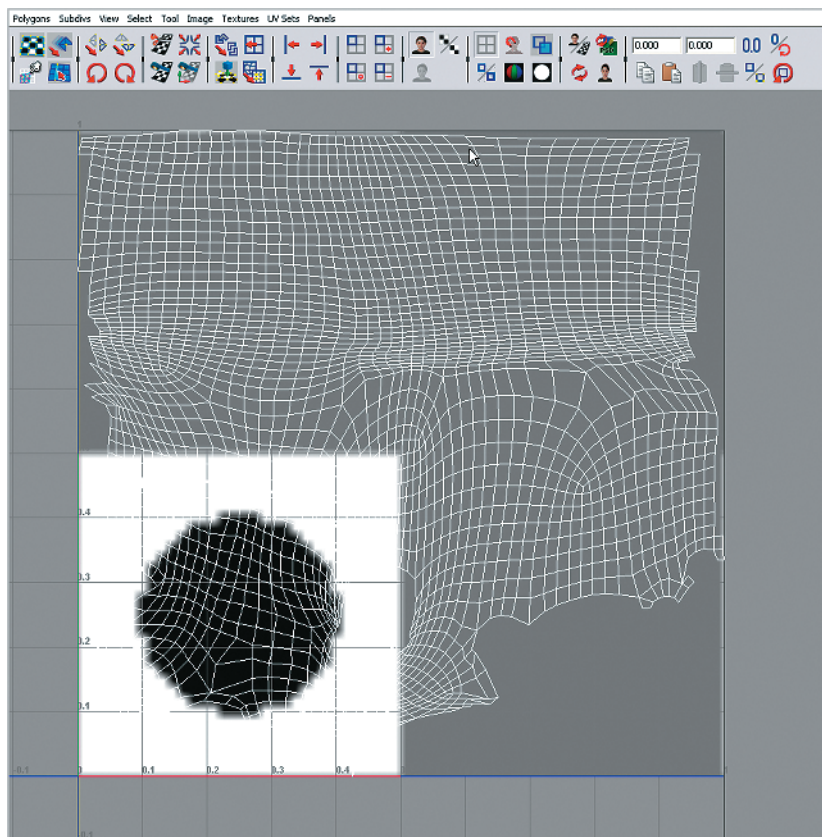


Figure 1.24: The UV Texture Editor with the circular ramp in the bottom left

13. Select the hand in the perspective window and then Shift+select the place2D texture node in the Hypershade. With the hand selected, the UVs in the UV Texture Editor won't disappear and we can see what's going on as we edit the attributes for the place2D texture node.
14. Make sure the Attribute Editor for the 2D texture node is open. In the **Translate Frame** fields, enter some numbers between 0 and 1 and watch what happens to the texture in the UV Texture Editor and the perspective window.
15. Try entering 0.5 for U and 0.35 for V. That's about where the splotch should be. If these values are not working for you, do some experimentation and try to get the dot to line up on the back of the wrist. Do a quick render of the perspective view, looking at the hand from the top.

We have a nice black dot where we want our splotch, but we also have a mostly gray hand (Figure 1.25).



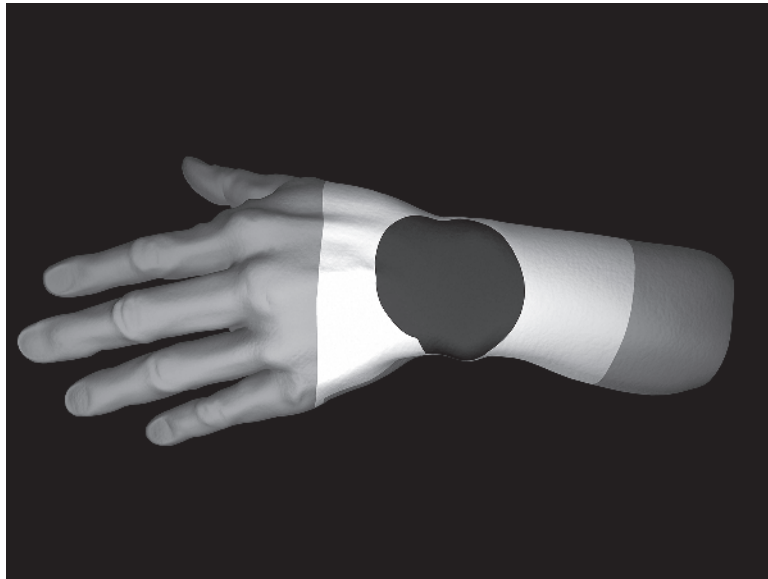


Figure 1.25:  
A gray hand with a  
minimalist tattoo

16. To get rid of the gray, we need to change the **Default Color** of the Ramp texture. This is the color the texture assumes from the outside of its borders on to infinity. When you lower the coverage of the texture, the **Default Color** becomes visible on the model. To change this, open the Attribute Editor of the ramp, and in the Color Balance folder, set the **Default Color** to white.
17. If you move the white marker on the ramp up and down, you'll see the dot grow and shrink. This is how we'll animate the growth of the infection.

Now we have a white hand with a black dot—looks like a mime caught a case of the black dot plague (Figure 1.26).

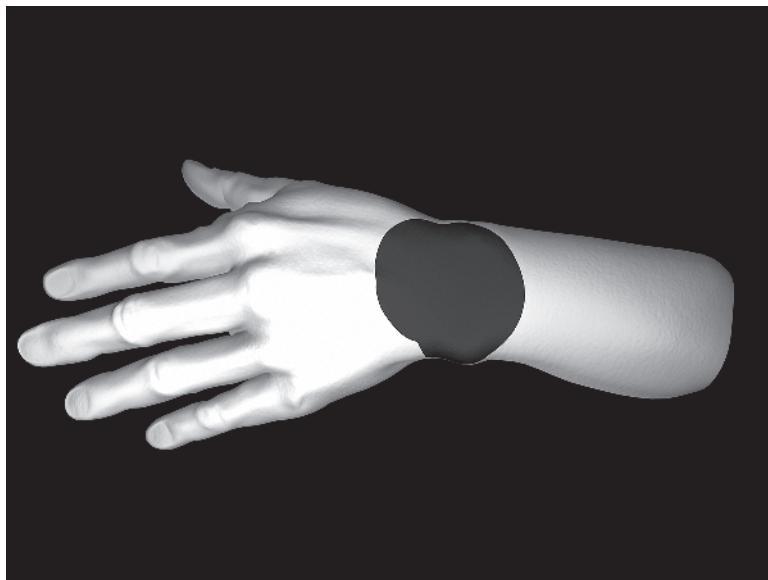


Figure 1.26:  
The hand with the  
infectionMask applied

## Creating a Splotch

These final steps will guide you through the process of making the texture look a bit more real and creepier.

1. To break up the dotlike shape, open the ramp in the Attribute Editor and move the Noise Slider up to about 0.2.
2. Set the Noise Frequency to 2. Do a quick render.

That's a nice splotch. As you move the white marker up and down, the splotch will grow and spread just like an infection. Those Ramp textures are handy!

3. Select the black color marker at the bottom of the ramp. In the Hypershade, switch to the Textures tab and find the psdFileTex3 node. With the Ramp texture in the Attribute Editor, middle-mouse-button-drag the icon for the psdFileTex3 node over the selected color for the black marker. When you release it, the texture will be applied to the black part of the ramp.
4. Select the hand model and open up the Relationship Editor for UV linking. Choose a texture-centric view.
5. You'll see the psdFileTex3 listed under Color. Select it and make sure that map1 is highlighted on the right side of the panel when psdFileTex3 is selected. Once you have verified this, move the camera above the hand, make sure the white color marker on the ramp is at halfway, and do a quick render. You should see the ugly black veins on the back of the hand. Move the white color marker down the ramp and render again. The veiny area should shrink.

This is one of the more important points of this tutorial. You can have multiple UV sets for different textures—even for textures that are connected to other textures. In other words, the Ramp texture is using the infection UV set and the Vein texture (psdFileTex3) is using map1 even though it is plugged into the color marker of the ramp. We could change it but we don't want to—this is exactly what we want.

## Bringing the Color Back

Now the ramp acts as a mask for the Vein texture and it's very easy to animate. Time to turn our diseased mime back into a zombie.

1. To get that fresh zombie color back into the hand model, we'll use a layered texture for the color channel. In the Hypershade, go to the Create menu and choose **Create > Layered Texture**. Open the layered texture's Attribute Editor.
2. From the Textures tab in the Hypershade window, middle-mouse-button-drag the psdFileTex1 texture to the right of the green box in the **Layered Texture Attributes**. A blue box will appear representing this texture in the layered texture (Figure 1.27).



Figure 1.27:  
The **Layered Texture Attributes**; the blue box represents psdFileTex1.

3. Click the check box under the green box to delete the green placeholder. Notice that when you hover your mouse pointer over these nondescript green and blue boxes, the name of the texture appears. This is one of the few ways you have to keep things straight with layered textures.

It's also quite easy to create extra layers by accident, either by double-clicking on the layered texture in the Hypershade or by clicking in the gray Layered Texture Attributes box in the Attribute Editor. This may cause your File textures to disappear on the model because they are being blocked by these new textures. If this happens, just delete the new green boxes.

4. Middle-mouse-button drag the infectionMask ramp from the Hypershade to the right of the blue box in the Layered Texture Attributes box.
5. Switch the order of these two textures in the Layered Texture Attributes window by middle-mouse-button-dragging the blue box representing the psdFileTex1 texture to the right of the blue box representing the infection ramp. (Boy, they could give us icons or something here just to make it clearer.)
6. Click on the blue box representing the infectionMask texture. In the attributes below, set **Blend Mode** to Multiply. You should see the psdFileTex1 icon appear in the Texture Sample 1 box above.

7. In the Hypershade work area, select the layered texture (you may have to hover the mouse pointer over the icon to see the name; right now it looks exactly like the psdFileTex1 texture), middle-mouse-button-drag it on top of the handSG shader, and choose **Color** from the pop-up window. This will automatically disconnect the ramp from the color channel and replace it with the layered texture. Do a quick render when it's done.
8. Now our zombie skin color is back, but he's got a black veiny texture growing on his wrist. Select the infectionMask ramp again, and in its Attribute Editor, switch **Interpolation** to **Spike**. This will soften the look of the area at the edges of the infection. Do a quick render—so that there is zombie juice flowing in his veins.
9. Animating this texture is so simple, the undead could do it! Open the Attribute Editor for the infection ramp. Select the white color marker and move it toward the bottom of the ramp to about position .015.
10. Make sure the Timeline is set to frame 1. Right-click over the field for the selected position for the white color and choose **Set Key**.
11. Move the Time Slider to frame 100. Move the white color marker all the way to the top and set another key on its selected position (Figure 1.28).


 You can try rendering the whole sequence or check out the QuickTime file on the CD.



Figure 1.28: The zombie hand complete with infection


## Abusing Ambient Occlusion: The Sci Fi Scanner

Our indefatigable art director has popped up with a new assignment. She would like to have a shot in which a character's skull is being scanned; we are to create a computer display for the scanning device. A ray of green light passes through the skull highlighting its contours in sort of a cross sectional fashion. Now, there are plenty of ways to create this effect but we're going to try something slightly wacky just to see what happens. We'll be repurposing mental ray's Ambient Occlusion node for something other than its intended function. (BWAH HA HA HA HA HA!)



The Ambient Occlusion texture is commonly used as a quick way to achieve the look of global illumination but without as much calculation. When the Ambient Occlusion node is connected to a shading network, the camera will shoot rays into the scene and test the region within a hemispherical area. If there is geometry within a certain distance of the surface with the occlusion texture applied, or if that part of the surface is occluded (blocked from view) by geometry, then the texture will make that part of the surface appear darker. This simulates the way that, in the real world, when two objects are very close to each other, a certain amount of ambient light is blocked and you get darkness near the edges where the two objects are close. It's kind of like a type of ambient light shadow.

However, we are not going for this particular look. Instead we're going to mess with the settings on the Ambient Occlusion node and take advantage of these properties to create a unique look.

1.  Open the skullScan\_v01.mb file from the Chapter 1 directory on the CD. Do a quick render from the perspective camera. In the view window, you may want to switch to wire frame just so you can see what's going on.
2. There's our skull with an eerie blue glow around it (Figure 1.29). If you look at a graph of the shader applied to the skull, you see that it has a pretty typical x-ray type shader applied to it. This shader uses the Surface Info node to control the positions of colors on a ramp. The ramp is then plugged into the incandescence of the shader. Parts of the geometry that are facing away from the camera at glancing angles will glow and parts that are more perpendicular to the camera are dark. Take a few minutes to examine how the shader works if you're not familiar with this technique.

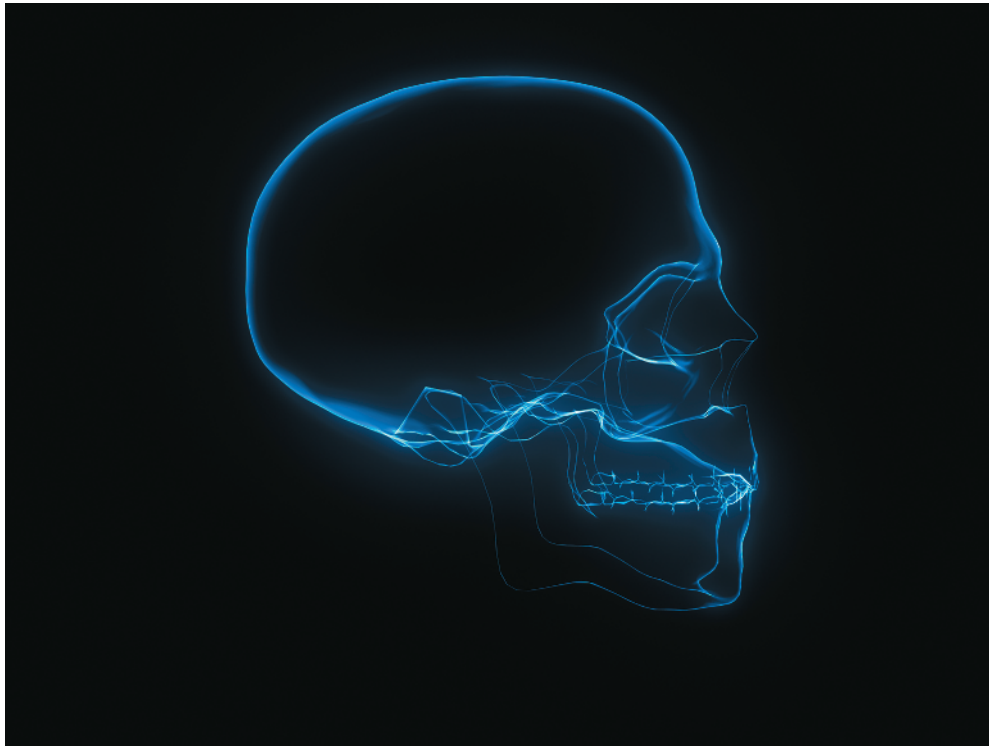


Figure 1.29: A rendering of the scene shows a skull model with fairly typical X-ray style shader applied.

3. Look in the Outliner and expand the scanner group. Along with the skull geometry, there is a hidden plane labeled “slice1.” Look in the Channel Box for slice1 and turn the visibility on for this object. If you scrub through the animation on the Timeline, you’ll see that the slice1 plane is keyframed so that it continually moves through the skull from front to back. This plane will become our scanner.
4. In the Render Settings window, switch to mental ray and select **Production** from the **Preset Render Settings** menu.

If mental ray does not appear in the list, open up the Plug-in Manager by choosing **Window > Settings/Preferences > Plug-in Manager**. In the list of plug-ins, make sure **Mayatomr.mll** is checked for loaded and auto-load. Now when you go back to the Render Settings window, you should see mental ray loaded.

So think about how the Ambient Occlusion node works. It’s essentially shading an object a certain way if geometry is close to it and shading it a different way if geometry is not close to it. The Ambient Occlusion node allows you to alter these parameters and change the distance and the intensity of the effect. Open the Hypershade, create a **mib\_amb\_occlusion** node, and take a look at its settings in the Attribute Editor (the **mib\_amb\_occlusion** node is found under the **Create > Mental Ray Textures** menu



in the Hypershade). Open the Attribute Editor for this node. Notice that there are sliders for the **Bright** and **Dark** colors. Who's to say we can't put a light color in the **Dark Color** channel and a dark color in the **Bright Color** channel? It's not like the Maya police are going to show up if we put a texture in these channels, either. I mean, the texture node buttons are right there next to the sliders just crying out to be messed with. OK, so let's mess with them.

### Creating the Scanner Shader

1. In the Outliner, expand the scanner group and select the object labeled "slice1."
2. Open the Hypershade, create a Lambert shader, and apply it to slice1.
3. Rename this shader "scannerShader."
4. Open the scannerShader in the Attribute Editor and set the **Transparency Color** to white so that the slice1 object is essentially invisible.
5. Open up the Hypershade and right-click over the scannerShader. Choose **Graph Network** from the pop-up marking menu.
6. Create an Ambient Occlusion texture (you can find this node listed under the Create mental ray textures list). If you've already created one, you can grab it from the Textures tab in the Hypershade and drag it down to the work area.
7. Middle-mouse-button-drag the mib\_amb\_occlusion texture over the scanner shader and choose **Incandescence** from the pop-up marking menu.
8. Open up the Ambient Occlusion texture in the Attribute Editor. Set the **Bright Color** to black and choose a mid to dark green for the **Dark Color** (0, 0.653, 0 for the RGB values works pretty well).
9. Set the Time Slider to 35 and do a test render.

That's pretty cool but a little intense. In the Attribute Editor for the Ambient Occlusion node, change **Max Distance** from 0 to .25. You might think that a 0 setting would cause it to sample no distance, but in fact it means that the entire scene is sampled. Setting **Max Distance** to .25 will tune the effect a little and give us more detail and less of a blown-out look (Figure 1.30).

You can also try adjusting the spread and the **Falloff** to tune the effect. **Spread** determines the size of the area above the point that is sampled. In this case, a small setting will look slightly brighter than a setting of 1. A lower **Falloff** setting will make the effect more intense for areas where geometry is close together and softer (or in our case, dimmer) for those areas that are farther apart. **Falloff** modifies the **MaxDistance** value so it has no effect unless there is a non-zero setting in **Max Distance**.

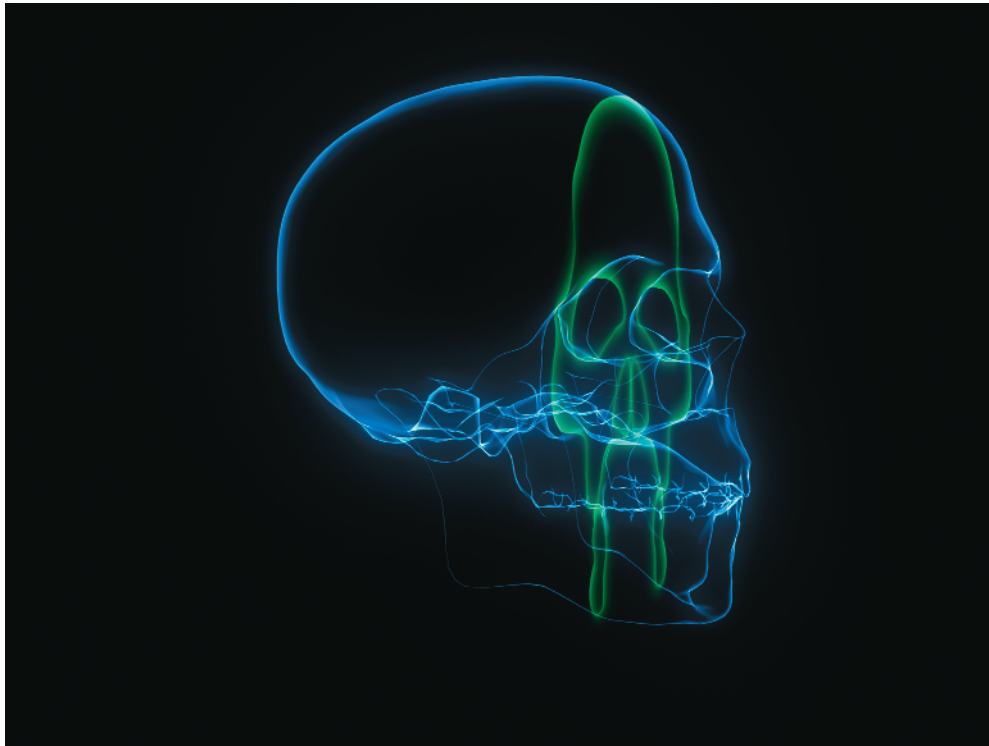



Figure 1.30: The plane intersecting the skull shows up when the Ambient Occlusion node is plugged in to incandescence.

## Tuning the Shader

1. In the Attribute Editor for the Ambient Occlusion node, hit the Texture button next to the **Dark Color** attribute (the attribute that you changed to green). In the texture panel, choose Grid.
2. In the attributes for the grid texture, change **Line Color** to a medium green and **Filler Color** to a dark green.
3. Set **U Width** to 0, **V Width** to 0.25, and **Contrast** to 0.85.
4. In the place2D Texture node for the grid texture, set **Repeat U Value** to 0 and **Repeat V Value** to 200 (Figure 1.31).
5.  Test render the frame now. Pretty cool. You can render out the whole animation or check out the QuickTime render skullScan.mov on the CD. For a finished example of the scene, open skullScan\_v02.mb in the Chapter 1 files on the CD.

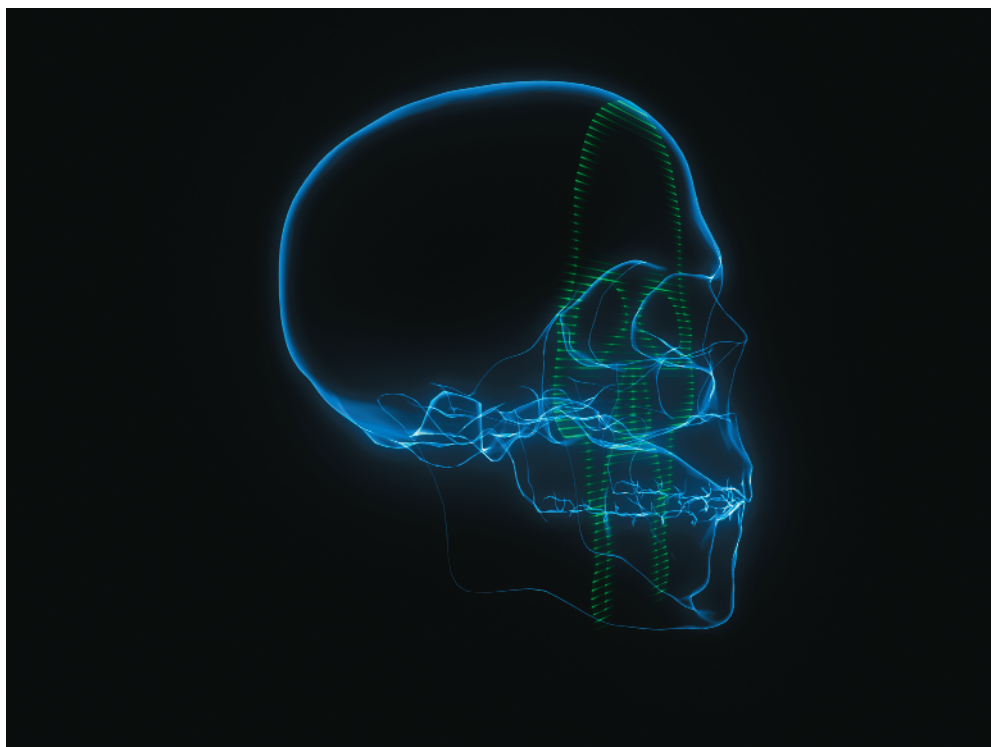


Figure 1.31: A grid texture has been added to the shader network to enhance the scanner look.

### Further Study

🌀 This technique has a wide range of applications. Try attaching the Ambient Occlusion texture to the incandescence of a shader applied to a wall. In the same way, set a light color for the **Dark Color** attribute and black for the **Bright Color** attribute. Place an invisible hand model near the wall for a very spooky ghostly hand print effect. Check out the `spookyHand.mb` file on the CD for an example (Figure 1.32).

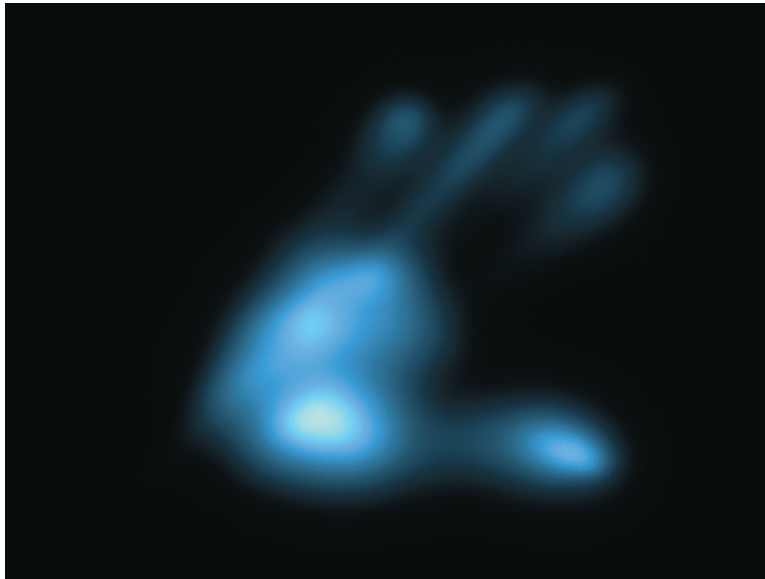


Figure 1.32:  
With the Ambient Occlusion node plugged into the incandescence of the wall shader, the hand print only appears when the hand is close to the wall. Mulder?

In Figure 1.33, the geometry behind the text has the same technique applied to its shader to create a glow behind the 3D text.



Figure 1.33:  
This same technique can be used to simulate backlighting for logos.

## Controlling Particles with Textures: Swimming Bacteria

This last exercise kind of segues to the next chapter on particles. Our art director has presented us with the problem of creating some swimming bacteria to be seen under a microscope. To make the bacteria look like they're alive and kicking, we'll use some ramps to control the direction in which the bacteria are moving. We'll be starting with a blank scene.



### Creating the Particle and Goals

Particle motion is commonly controlled through the use of fields, goals, and expressions, or any combination of the three. Almost anything can be a goal for a particle object, even other particles. In this exercise we'll see what happens when two particle objects become each other's goal object.

1. Start a new file, switch to the Dynamics menu set, and create two emitter objects by choosing **Particles > Create Emitter** twice.
2. Select each emitter and move them apart at least 10 units on the grid.
3. Go into the Attribute Editor for each emitter and set Type to **Omni**, set **Rate** to 25 particles per second, and make sure **Speed** is at 1.
4. We're going to make a goal for both particles. A goal is like a magnet that attracts the particles; the strength of the magnet is dependent on the goal weight. In this case, we are going to make each particle a goal for the other particle and vice versa. In other words, each individual particle will attract a particle from the other emitter and that particle will attract its opposite. Should be interesting. In the Outliner, select particle1 and then Control+select particle2. Choose **Particles > Goal** and open the options.
5. In the options, set **Goal Weight** to 1.0 and make sure that **Use Transforms As Goal** is off. If it is on, then the center of the particle object would be the goal instead of the individual particles and that's not what we want. Click Create to apply these settings.
6. Now select particle2, Control+select particle1, and choose **Particles > Goal** to apply the same settings for particle2.
7. Make sure the Time Slider is at least 300 frames and play the animation. Wow, that's not very interesting.
8. The **Goal Weight** for each particle is set to 1, so the particles reach each other in almost no time. In the Channel Box for the particle1 and particle2 shapes, scroll down and set **Goal Weight** for each particle to 0.5. Play the animation and see what happens.

OK, that's slightly more interesting, but not very bacterial. Let's see how we can control the goal weights using some Ramp textures.

## Creating the Ramps

The goal weight for the particles will be assigned by the values on a Ramp texture. The initial values will be derived by assigning the goal weight to a random position along the ramp, and from there the weight value will move up the ramp as time goes by. As the goal weight moves through bands of different colors, the value will change. This has some advantages over creating an expression for the goal weight in that it's easier to set up and easier to tune when you need to adjust the animation. However, a few simple expressions will be involved in the setup. First make sure the goal weights for the two particle objects have been set back to 1.0 and proceed with the following steps:

1. Open up the Attribute Editor for particleShape1. In the **Emission Attributes**, set the **Max Count** value for the particle object to 200.
2. We need an attribute that will give us a random initial placement along the ramp for the goal weight of each particle. To do this, we'll create a simple custom attribute. Under the Add Dynamic Attributes rollout, click the General button.
3. In the Add Attribute panel, type **startWeight** for the attribute name.
4. Set **Data Type** to Float (meaning a number that can have a decimal value), and set **Attribute Type** to Per Particle (Array) so that each particle can have a different value assigned. Click the OK button to add the attribute (Figure 1.34).

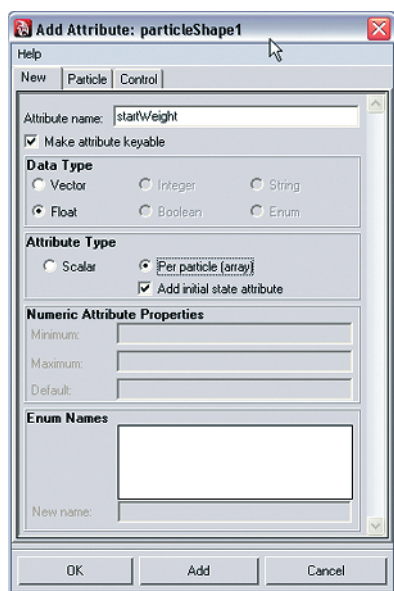


Figure 1.34: Creating the **startWeight** attribute

5. In the Per Particle Array Attributes list, you should see the new **startWeight** attribute. If it does not appear, click the Load Attributes button at the bottom of the panel.
6. In the blank slot next to the **startWeight** attribute, right-click and choose **Creation Expression**. In the Expression Editor's main box, type the following expression:

```
seed(particleId);
particleShape1.startWeight=rand(0,.5);
```



7. Click the Create button to create the expression. If you get an error message, double-check and make sure you typed it in correctly. This expression just says that as each particle is created (it's a creation expression), it is born with its **startWeight** attribute set to a random value from 0 to .5 (Figure 1.35). The seed attribute ensures that the seed value for the random number generator is consistent so that you'll get the same random values generated each time you open the file. This helps especially when sharing the file with other animators. In this case, the **seed** value is set to the ID number of each individual particle.

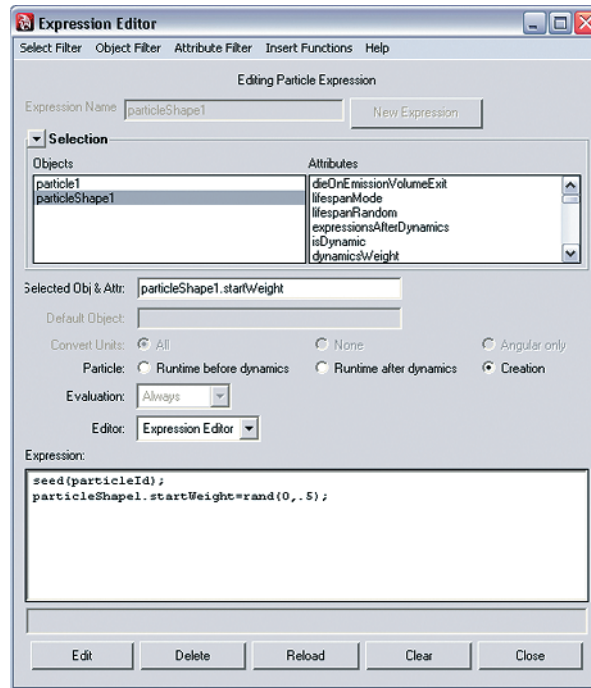


Figure 1.35:  
The creation expression for  
the **startWeight** attribute

8. Next we'll create the Ramp texture for the goal weight. In the space next to the **goalPP** attribute in the Per Particle Array Attribute list, right-click and choose **Create Ramp** from the pop-up menu. Open the options.
9. In the options, set the **inputV** to **startWeight**.
10. Click OK to add the ramp.
11. Right-click in the same slot again and choose **Edit Ramp**. Edit the ramp so that there are bands of gray and dark gray (almost black) up and down the Ramp texture (Figure 1.36).

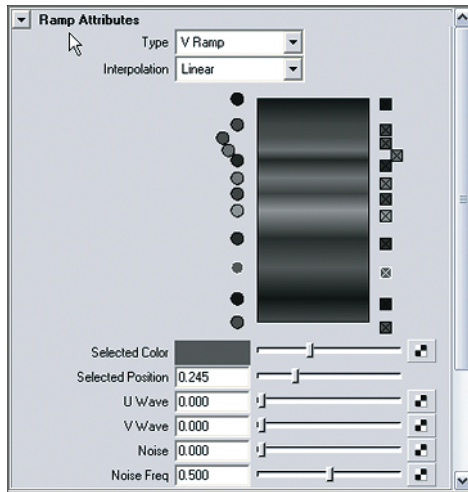


Figure 1.36:  
The ramp has a series of gray and dark gray bands; these will determine the value for each particle's goal weight.

12. Go back and repeat steps 1 through 11 for particle2. When you're done, play the animation and note the movement of the particles. Make the goalPP ramp for particleShape2 different from the ramp used for particleShape1.

It's kinda neat but not quite germey enough. The particles are moving about and attracting each other randomly, but it needs something more. First let's see what we can do about the goal weights. It's obvious from the fact that different particles move at different rates that the individual particles have different weights. We can even see what the rates are by changing the particle type.

## Adjusting the Ramps

We can see the goal weight value for each particle by switching the particle's Render Type to "numeric". This will help us as we adjust the ramp to create a more natural movement.

1. Under the particle Render Type for particle1, set the type to **Numeric**.
2. Click the Add Attributes For Current Render Type button.
3. In the Attribute Name field, replace "particleID" with "goalPP."
4. Play the animation and try to follow the numbers (Figure 1.37). The higher the number, the faster the particle. These numbers are derived from the values of the colors on the ramp attached to the **goalPP** attribute. You can make sure that these numbers are coming from the ramp by going to the Ramp texture assigned to **goalPP** and deleting all the colors except one. When you play the animation, you'll see that all of the goal weights are now equivalent to the value for the color of that last remaining marker on the map. Actually, since the **startWeightRampPos** value was set from 0 to 0.5, these values are really coming from the lower part of the ramp. There's a reason for doing this.

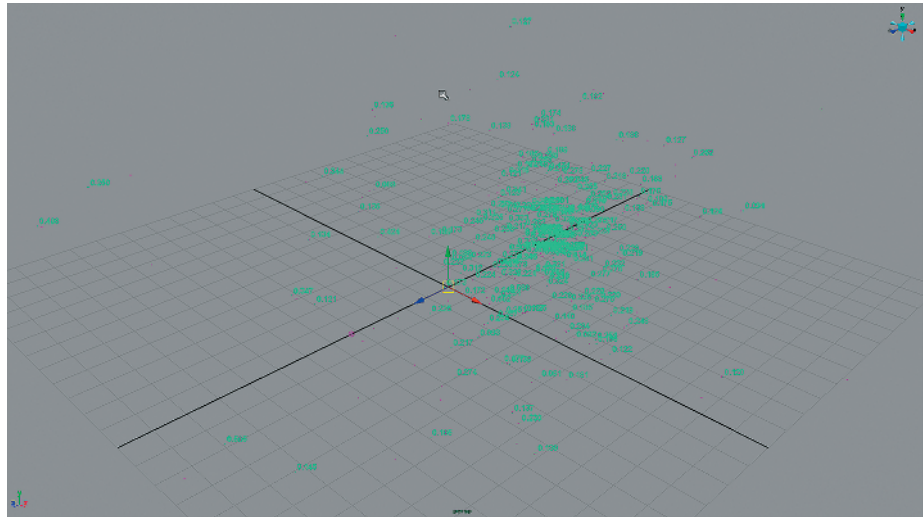


Figure 1.37: The particles set to the numeric render type. The goal weight values for each particle can now be seen.

5. We want to make it so that as time goes on, the goal weight for each particle changes randomly. We can do this by adding another simple expression. Open the Attribute Editor for particleShape1. Right-click over the slot next to the Start Weight value. Choose **Runtime Before Dynamics**.
6. In the Expression box type this:
 

```
particleShape1.startWeight= particleShape1.startWeight+.01;
```
7. With this expression added to each frame, the **startWeightRampPos** value will move up the ramp in increments of .01. Play the animation and watch the numeric values move up. The numbers will change over time according to the values of the colors in the bands on the ramp. Add this same expression to the **startWeight** attribute of particleShape2.
8. It's kind of cool, there's a springy back and forth motion to the movement of the particles. It needs just a little more to keep the motion random. Does this mean adding more expressions and ramps? Nah, just add a turbulence field. Select particle1 and choose **Fields > Turbulence**. In the options set **Attenuation** to 0 and **Magnitude** to 10.
9. Do the same for particle2 so that each particle has its own turbulence field (Figure 1.38).

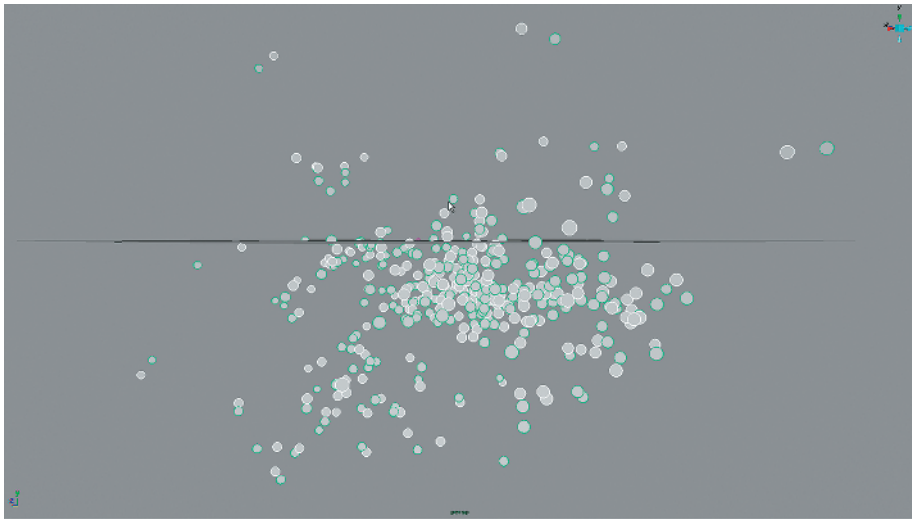


Figure 1.38: The particles set to the blobby surface render type. The ramps on the goal weights make them appear as though they are swimming bacteria.

Now when you play the animation, things start to look germey, in a good way!

### Further Study

Applying ramps to per-particle goals is just the beginning. You can use similar techniques on the **rampPosition**, **rampVelocity**, and **rampAcceleration** values as well as your own custom attributes. Here are some suggestions for further tuning the scene you have just created.

1. In the Array Mapper for the ramps applied to the **goalPP** for each particle, set the **Max Value** to 0.5. This will restrict the value range so that none of the weights are higher than 0.5. This may slow down the faster particles. You can also set the array mapper's **Max Values** to different numbers for the ramps applied to each particle just to see what happens. Alternatively, you can adjust the color gain on the ramps to bring the upper limit of the colors down or raise the color offset to boost the lower range.
2. Lower the **Conserve** value for the particles. A setting of 0.8 for each will lead to some wild spiral effects.
3. Randomize the phase values for the turbulence values. To do this, open the Attribute Editor for **turbulenceField1**, right-click over the **PhaseX** value, and add an expression. The expression `turbulenceField1.phaseX=noise(time);` is a good one.
4. As we all know, salmonella propels itself through the cytoplasm of a cell by building a tail made of actin filaments; they tend to look like little jets. You didn't know that? Hey, all the cool kids know their microbiology. In any case, you can simulate this effect by turning the **particle1** and **particle2** objects into emitters and then setting a very low value for each emitted particle's life span.
5. To create the look of the bacteria, you can try particle instancing with a modeled object, or if the bacteria are going to be very small, using blobby surfaces works quite well.