Recipes and Tutorials

Getting Started

RenderMan Features

RfM Pro

Tips and Tricks

Recipes

Videos

Navigate using the menus on the left.

○ RfM Fast Start

Getting Started

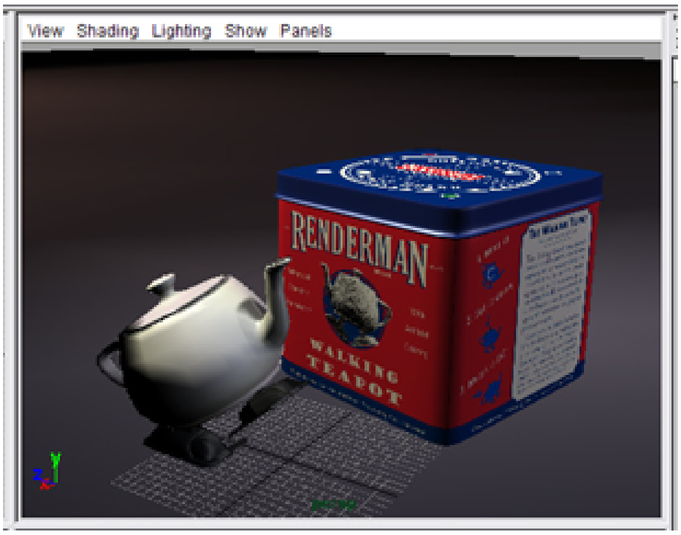
Rendering a Picture

To get started, let's render the Maya scene, *teapot_and _box.ma*.
(Where are the tutorial files?)

1 THE MAYA SCENE

First of all, make sure that RenderMan is properly set up.

With RenderMan for Maya fully loaded, open up the Maya scene, *teapot_and _box.ma*. This scene is simple, but it contains a number of Maya Materials, including the ramp shader and some texture maps. The teapot is a hierarchical subdivision surface. The box is constructed from NURBS. The scene should look something like this:



Maya Scene

2 RENDERING WITH RENDERMAN

First, render the scene using the Maya renderer:

```
Render-> Render Current Frame
```

The Maya render should look like this:

The Maya Renderer

Next we'll render the scene using RenderMan.

First, choose RenderMan as the renderer:

```
        Render-> Render Using-> RenderMan
```

Now you're ready to render the scene with RenderMan!

```
        Render-> Render Current Frame
```

With any luck, your result should look like this:

Rendered with RenderMan

Congratulations! You've rendered an image with RenderMan!

3 SUMMARY

RenderMan for Maya allows Maya scenes to be rendered with RenderMan simply by switching renderers. No additional set up is required. RenderMan for Maya will translate the majority of elements in any given Maya scene file. This allows the Maya artist to harness the full power of RenderMan with a minimal amount of hassle.

You may notice, however, that in the example above both images render in about the same amount of time and with approximately the same level of quality. As you become more familiar with RenderMan you will become aquainted with where its strengths are to be found. For instance, RenderMan is memory efficient and it excels at rendering lots of geometry. In the example above, if we were renderering 100 teapots, and not just one, we would begin to appreciate RenderMan's ability to handle large, complicated scenes.

Learn more about some of the other **fundamental strengths** of RenderMan:

Motion Blur

Depth of Field

Displacements

Ray Tracing

Particles

Fur and Hair

RenderMan also has **advanced features** for creating sophisticated effects:

Deep Shadows

Global Illumination

Subsurface Scattering

AOVs

A familiarity with RenderMan will help to successfully leverage its strengths in the production of high quality CG imagery.

Prev | Next

Prev | Next

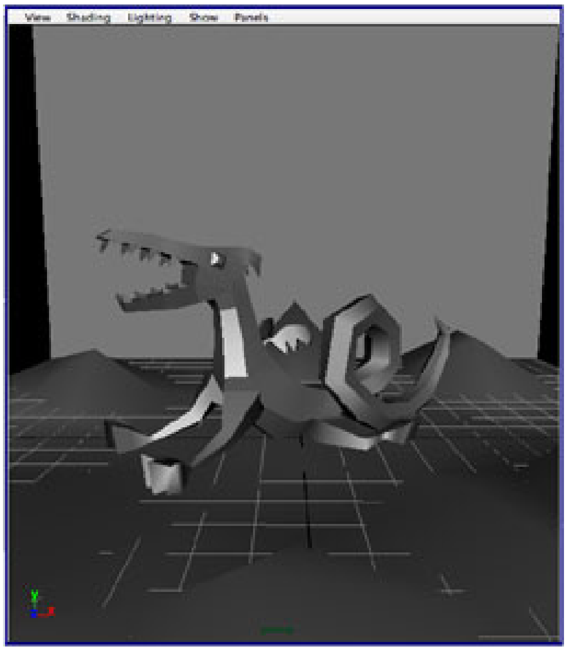
Ray Tracing

Fot this tutorial open the Maya scene, *dragon.ma*.
(Where are the tutorial files?)

1 THE MAYA SCENE

In this tutorial we will take a look at ray-traced reflections and refractions. Most familiar ray tracing operations are easy to set up with RenderMan for Maya. For instance, basic ray-traced reflections, refractions, and shadows can be created using the standard Maya methods. Other advanced ray tracing effects, like Global Illumination, require special RenderMan workflows.

Open up the Maya scene, *dragon.ma*. This scene is simple, with a subdivision surface dragon and a NURBS plane inside a polygonal Cornell Box , a couple of Maya Materials, and a couple of lights. The scene should look something like this:



Maya Scene

First things first: if you look at the Maya materials and what they have been attached to, you might notice that there's not much to reflect and/or refract in this scene. That's easily fixed, of course. Select the Grid material (it's just a Lambert) and let's map a simple image of a grid to the color. The image is located in the sourceimages folder of the `rfm_project` (`~/rfm_project/sourceimages/grid.tif`).

2 RAY-TRACED REFLECTIONS

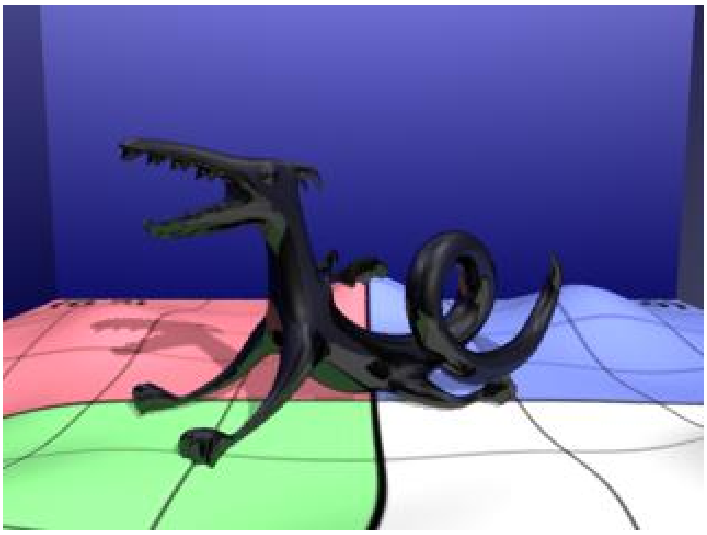
First we'll add ray-traced reflections to the model. The first step is enabling ray tracing, which is done via the Render Globals. Open the Render Globals and select the *Features* tab, then click the Ray Tracing box in the *Ray Tracing* section to enable ray tracing.

● **Note**

RenderMan is a hybrid scanline renderer which has an advanced ray tracing sub-system. Enabling ray-traced effects requires more memory resources than pure scanline rendering. Generally ray tracing should be used when these effects are absolutely necessary and cannot be created (faked) in another way.

Next we'll create a ray-traced chrome material. Start by assigning a new Blinn to the dragon. Set your Blinn color to something kinda chromey (or brassy, depending on your metallic preferences). You can leave the specular settings at their default levels, except the Reflectivity, which you should crank all the way up to **1.00**. Some folks like to match the specular color to the material color, but we wouldn't force your hand. Then pull back on the Diffuse, all the way, if you're so inclined.

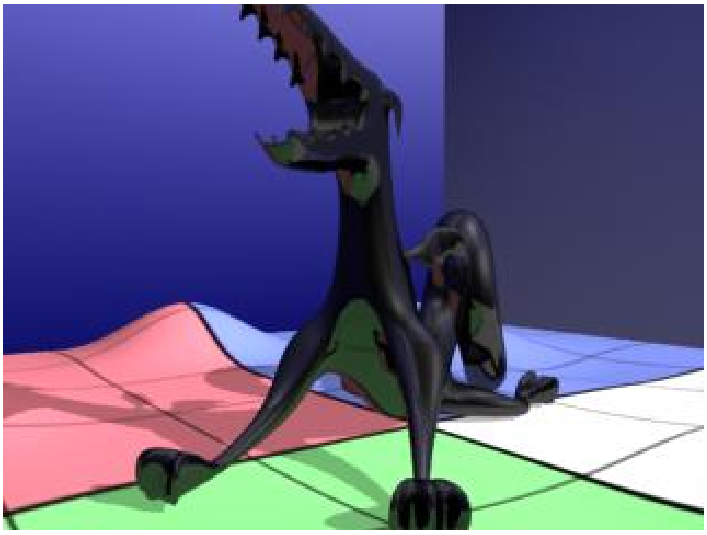
Render! Your rendered image should bear some resemblance to the image below:



Ray-traced Reflections

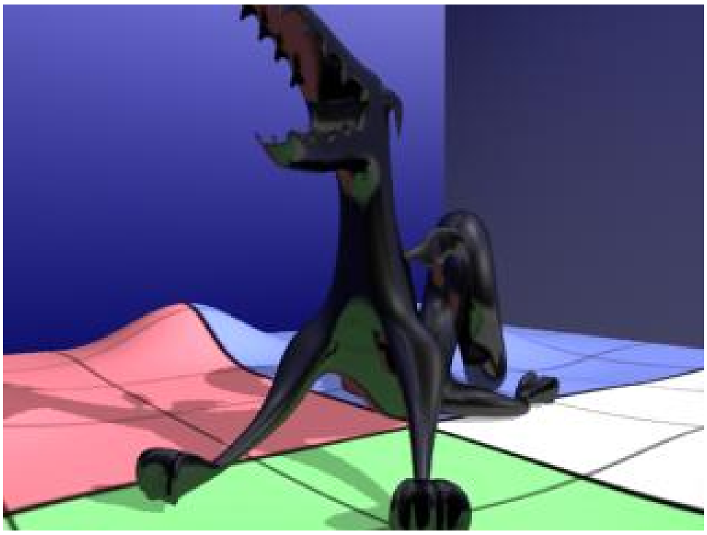
That's the basic approach, but RenderMan for Maya provides some added control over your ray-traced reflections. Go back to your Blinn in the Attribute Editor, then go to the Attributes menu and select RenderMan-> Add Reflection Controls. This will create a new section of Extra RenderMan Attributes that give you control over the quality and appearance of your reflections.

First thing we do, let's get in closer to get a better look at the reflections, then let's increase the Reflection Samples to improve the overall quality of the ray-traced reflections. 6 seems like as good a place to start as any. Rerender, and note the pleasant improvement in the image. Mmm, crispy ⟩



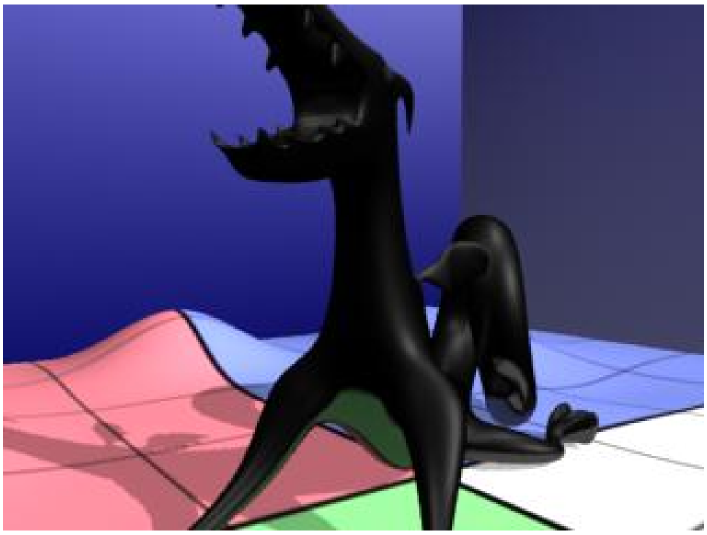
Ray-traced Reflections; Reflection Samples: 6

Perhaps, in fact, that's a bit too crisp for us. Luckily, we can play with the reflection blur, too. Crank that Blur setting all the way to 1.000 and rerender. Those edges have softened up quite a bit now.



Ray-traced Reflections; Blur: 1

RenderMan for Maya also gives you two ways to limit what you see in your reflections: you can create a Maya set and specify it, or you can manipulate the Reflection Max Dist parameter. Our scene's a bit simple to warrant using a Maya set, but let's play with the Max Dist. It's set by default to something like a bazillion; let's lower that to 2 , and rerender. Only elements of the scene within 2 units of our reflective surface will be visible in our reflections, see:



Ray-traced Reflections; Reflection Max Dist: 2

The blue from the Cornell Box is no longer visible, and we cannot see the grid at the higher points of the dragon.

3 RAY-TRACED REFRACTIONS

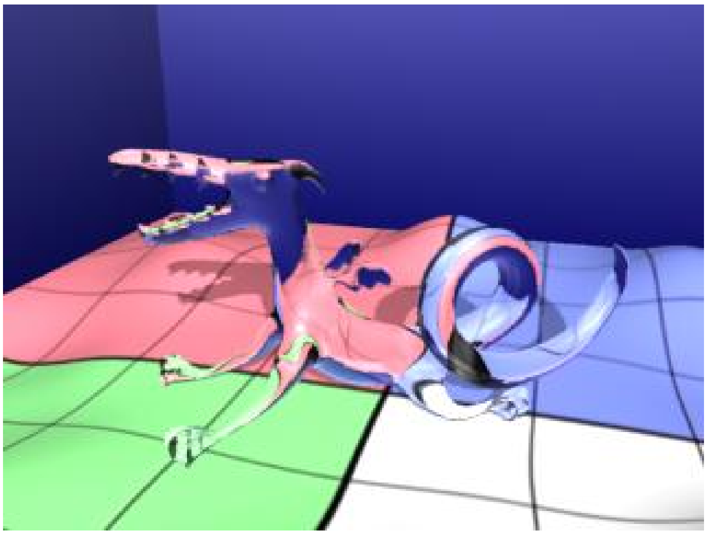
Light refracts through transparent surfaces, such as glass. Ray tracing allows you to re-create this effect with ease. In fact, ray-traced refractions are just a couple of simple tweaks to the Blinn material in the scene we've been working with.

First things first: let's reposition our camera to get a better view of the refractions, up and to the right a bit, so that we're looking down at the grid, through the dragon.

Next, open the Attribute Editor for our Blinn. Typically we are told to change our Color attribute to Black for a glass-like material, but we're also going to spike the Transparency, dragging that slider all the way to the right, so go figure) For now, go ahead and drag the Color slider all the way to the left and the Transparency slider all the way to the right.

In the *Raytrace Options*, check the Refractions checkbox, thereby enabling refractions. In the real world glass has a refractive index 'round about 1.5, crystal 'round about 2.0. For our purposes, a crystal dragon will do a slightly better job of showing off the effect, so go ahead and increase the Refractive Index to **2.000**. And, what the heck, since we're splurging on crystal, we might as well make it heavy crystal go ahead crank the Surface Thickness up to **1.000**.

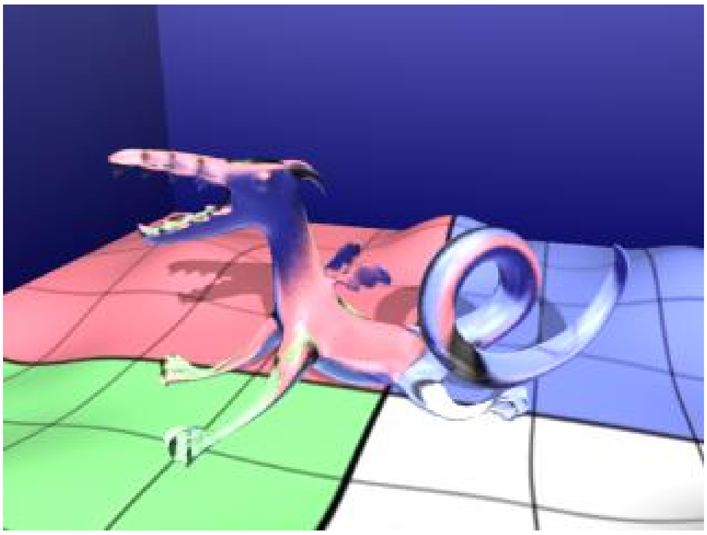
Do that rendering thing again, and you should have something that looks like this:



Ray-traced Refractions

Pretty neat, eh?

Once again, you can Add Refraction Controls the same way you added Reflection Controls. Feel free to play with the various parameters to see their effect. Note the correlation between speed and quality, and keep that in mind as you work on your own projects.



Ray-traced Refractions; Refraction Samples: 6 and Blur: 1

4 SUMMARY

As we've seen in these two simple exercises, ray tracing with RenderMan for Maya is fast and easy. Once you're comfortable, check out the RenderMan for Maya tutorial on Global Illumination. Here's more information about Reflections and Refractions.

Prev | Next

Motion Blur

To get started, let's open the Maya scene, *motion_blur.ma*.
(Where are the tutorial files?)

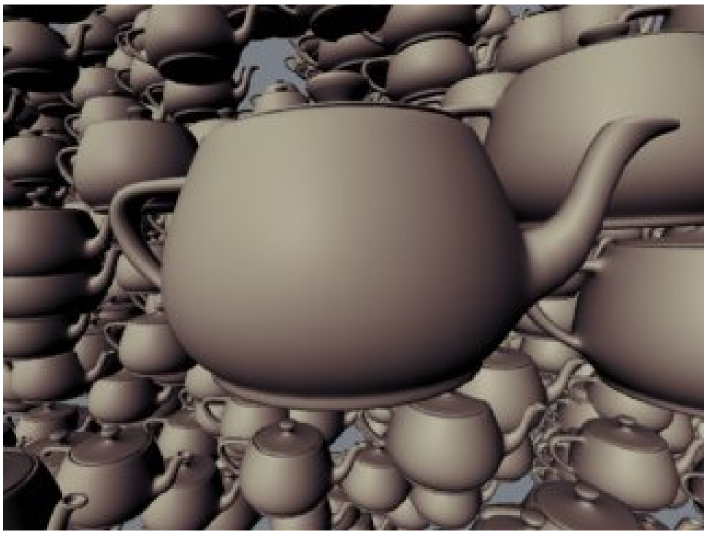
1 2,000 TEAPOTS

Open the scene, *motion_blur.ma*, in Maya. For this tutorial we have around 2,000 teapots flying through the air. Note that this was accomplished using Maya's *Particle Instancing* feature, which RenderMan For Maya supports. Now 2,000 teapots is more teapots than anyone really needs, but it will serve well as a little stress test for RenderMan's motion blur.

First render the Maya scene without motion blur. Make sure your renderer is set to RenderMan and run your particles up to frame 35 or so, then render.

```
Render-> Render Current Frame
```

You should get an image like the one below:



Flying teapots
(Rendering time: 12 seconds)

2 ENABLING MOTION BLUR

Motion blur would be a nice effect to add to these high-velocity teapots. We can add motion blur by opening the Render Globals and selecting the *Features* tab. Enable motion blur by clicking the appropriate check box (next to *Motion Blur*). Now Render the Maya scene:

```
Render-> Render Current Frame
```

You should get an image like the one below:

Motion blur enabled
(Rendering time: 19 seconds)

3 SHUTTER ANGLE

An important motion blur control is *Shutter Angle*. Shutter Angle controls how blurry the motion blur is going to be. The default value is "80". A value of "360" means that the shutter will be open the entire frame, creating very blurry motion blur. A value of "1" means that the shutter will be open 1/360 of the frame, and there will be little noticeable blur at all.

For our image, we could use a little more blurring. Increase the Shutter Angle to "180" and render again.

```
    Render-> Render Current Frame
```

You should get an image like the one below:

Motion blur with a shutter angle of "180"
(Rendering time: 36 seconds)

4 ADJUSTING PIXEL SAMPLES

Pixel Samples is the most important setting when rendering motion blur, especially the extreme motion blur in this example. In the images above the motion blur is rather grainy , which is undesirable. By increasing the Pixel Samples we create smoother motion blur. We can do this by opening the Render Globals and selecting the *Quality* tab. The Pixel Samples setting defaults to 3x3. Increase the Pixel Samples to 8x8 (which will cause RenderMan to sample each pixel many more times). Now render the scene again:

```
Render-> Render Current Frame
```

You should get an image with much smoother blur:

Motion blur with Pixel Samples of "8"
(Rendering time: 46 seconds)

5 SUMMARY

RenderMan handles motion blur well, especially for large scenes with lots of fast-moving objects. By increasing the Pixel Samples, higher quality blur can be created, but, to avoid excessive render cycles, Pixel Samples should only be increased as much as is required.

For more information about configuring motion blur see the Features tab of the Render Globals. Read more about *Pixel Samples* in the Quality tab of the Render Globals.

Prev | Next

Prev | Next

Depth of Field

To get started, let's open the Maya scene, *depth_of_field.ma*.
(Where are the tutorial files?)

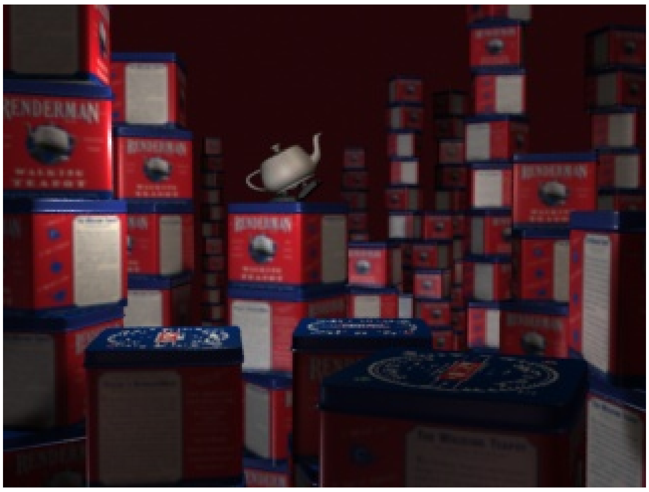
1 ANALYZING THE SCENE

Open the scene, *depth_of_field.ma*, in Maya. For this tutorial we have numerous stacks of boxes piled up around the scene. We will use depth of field to create a focal point for the scene. RenderMan uses an advanced 3D depth of field that is quite accurate. RenderMan's depth of field is *not* a post-process effect. Expect RenderMan to give you depth of field that has accurate anti-aliasing and placement.

First render the Maya scene without depth of field. Make sure your renderer is set to RenderMan.

```
Render-> Render Current Frame
```

You should get an image like the one below:



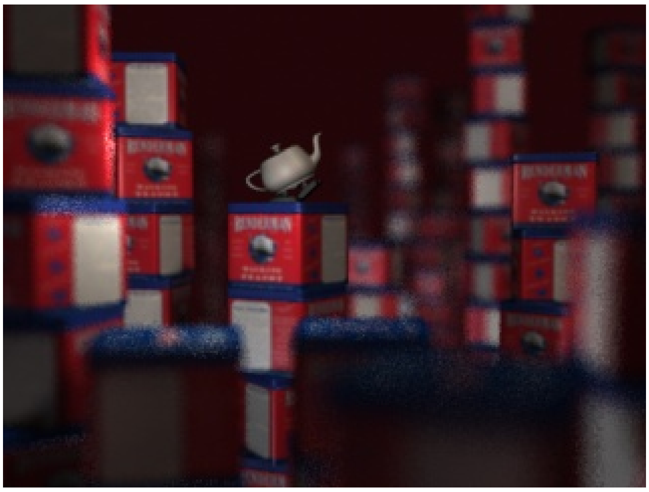
Stacks of boxes
(Rendering time: 9 seconds)

2 ENABLING DEPTH OF FIELD

Next we'll enable depth of field. We can enable depth of field simply by using Maya's depth of field controls. Just open the main camera in the Attribute Editor and enable the effect by clicking on the *Depth of Field* check-box. Now Render the Maya scene:

```
Render-> Render Current Frame
```

You should get an image like the one below:



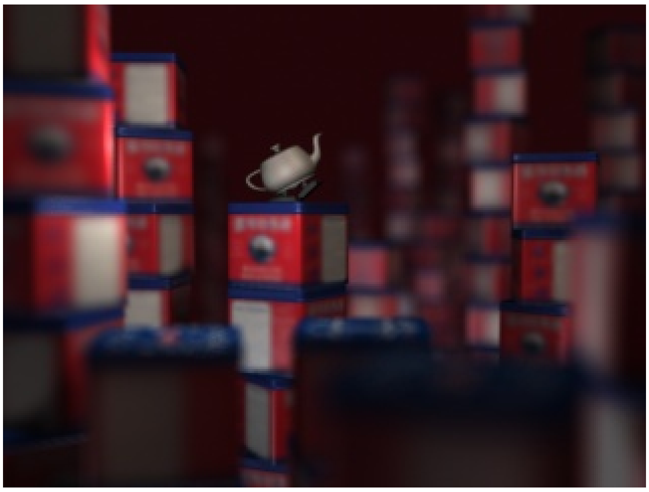
Depth of field enabled
(Rendering time: 19 seconds)

3 ADJUSTING PIXEL SAMPLES

For RenderMan, *Pixel Samples* is the most important quality setting for depth of field, especially extremely blurry depth of field. In the image above the depth of field is rather grainy , which is undesirable. By increasing the pixel samples we can create a smoother effect. We can increase Pixel Samples by opening the Render Globals and selecting the *Quality* tab. The Pixel Samples setting defaults to 3x3. Increase the Pixel Samples to 12x12 (which will cause RenderMan to sample each pixel many more times). Now render the scene again:

```
Render-> Render Current Frame
```

You should get an image with much smoother depth of field:



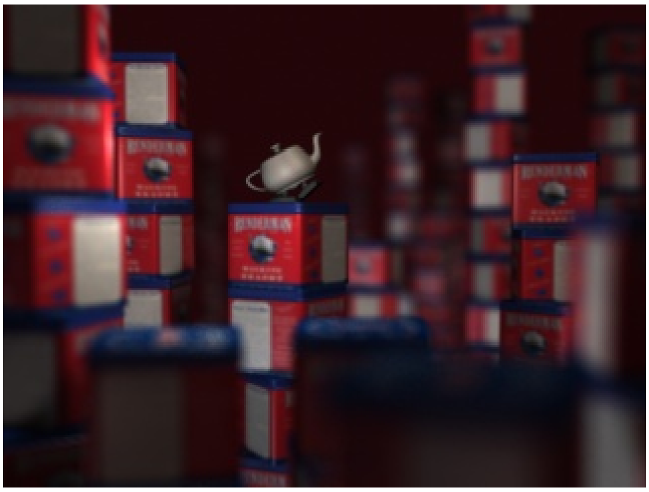
Depth of field with Pixel Samples of "12x12"
(Rendering time: 27 seconds)

4 ADJUSTING MOTION FACTOR

Motion Factor is a control used to render motion blurred objects more efficiently. Motion factor uses a lower quality shading rate on motion blurred objects, assuming (correctly) that if stuff is moving, it needs less detail. Motion blur and depth of field, however, are computed in very similar ways, and, in fact, motion factor affects objects with depth of field, too. In this case, we don't want any motion factor optimizations. To fix this, we'll set motion factor to "0" and the textures on the boxes will retain their detail. To set motion factor to zero, open the *Features* tab of the Render Globals and set Motion Factor to "0". Now render the scene again:

```
        Render-> Render Current Frame
```

You should get an image with more detail in the textures. You'll notice the text is clearer:



Depth of field with Motion Factor of "0"
(Rendering time: 41 seconds)

In certain cases you'll combine motion blur and depth of field in the same scene. In these cases some (stationary) objects require a Motion Factor of zero, while other (fast moving) objects can benefit from a higher Motion Factor. In such situations you may explicitly add the Motion Factor attribute to individual surfaces, so any surface can have its own Motion Factor setting. To add attributes, select the surface and add the Motion Factor attribute from Maya's attribute editor.

```
        1) Attribute Editor: Attributes-> RenderMan-> Manage Attributes
        2) In the pop-up window, select and add motion factor.
        3) Motion factor will now appear under the Extra RenderMan Attributes sub-tab, ready
for adjustment.
```

6 SUMMARY

RenderMan renders true 3D depth of field, which ensures correct anti-aliasing and positioning of the depth of field effect. It is not a post-process effect. By increasing the Pixel Samples, higher quality depth of field can be created, but, to avoid excessive render cycles, Pixel Samples should only be increased as much as is required.

For more information about configuring motion blur see the Features tab of the Render Globals.
Read more about *Pixel Samples* in the Quality tab of the Render Globals.

Prev | Next

Displacements

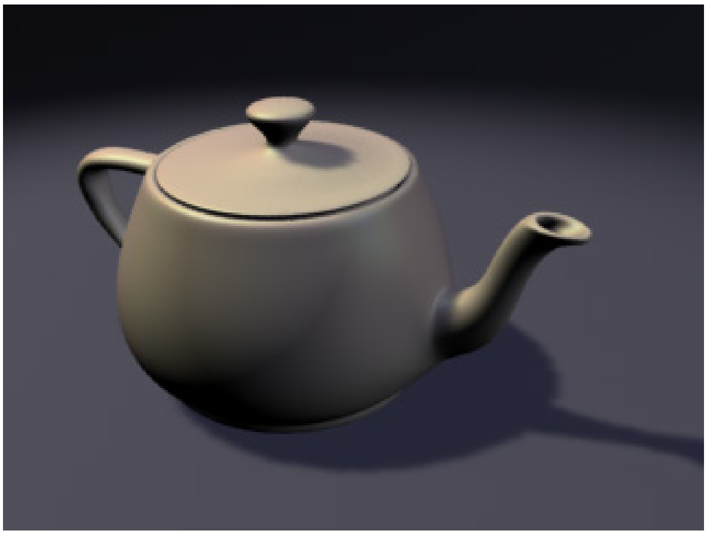
To get started, let's open the Maya scene, *displacements.ma*.
(Where are the tutorial files?)

1 WITHOUT DISPLACEMENTS

Render the Maya scene, *displacements.ma*, with RenderMan.

```
Render-> Render Current Frame
```

You should get an image like the one below:



Maya Scene

(Rendering time: 3 seconds)

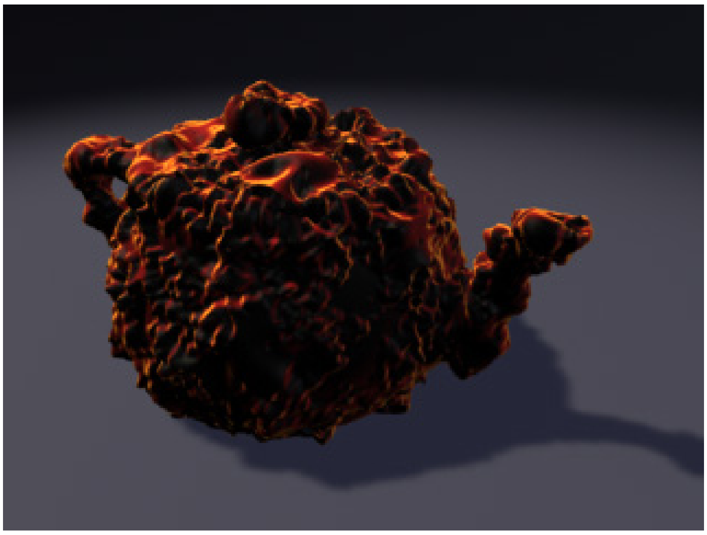
2 ATTACHING A DISPLACEMENT SHADER

Next attach the Maya Material, *funky_displacement*, to the teapot. This Material network has already been constructed for you and can be found in the Maya HyperShade.

Once you've attached the displacement shader, render the scene again.

```
Render-> Render Current Frame
```

Now you should get an image with some nice displacement, like this:



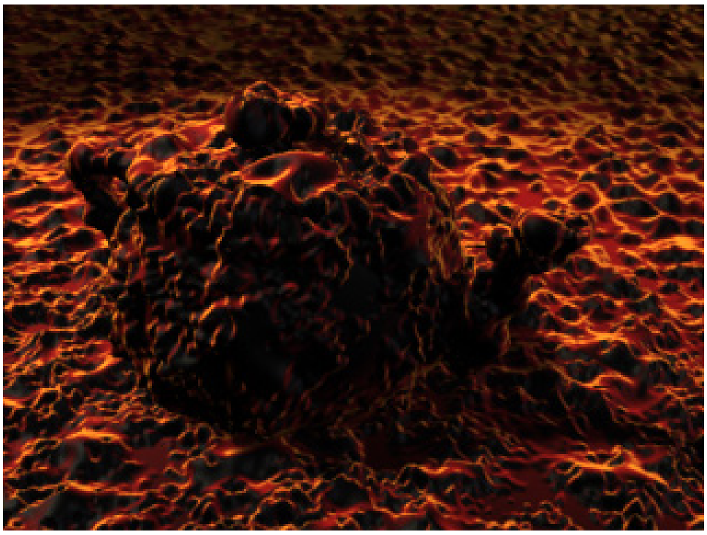
Teapot with Displacement Shader

(Rendering time: 6 seconds)

Notice the amount of detail that the displacement shader can quickly create on very basic geometry   even the shadow is displaced. But we're not going to stop here. Next attach the Maya Material, *funky_displacement*, to the ground. When you've attached the displacement shader, render the scene again.

```
        Render-> Render Current Frame
```

Now we've got a lot of displacement, as we see in the image below.



Entire scene with Displacement Shader

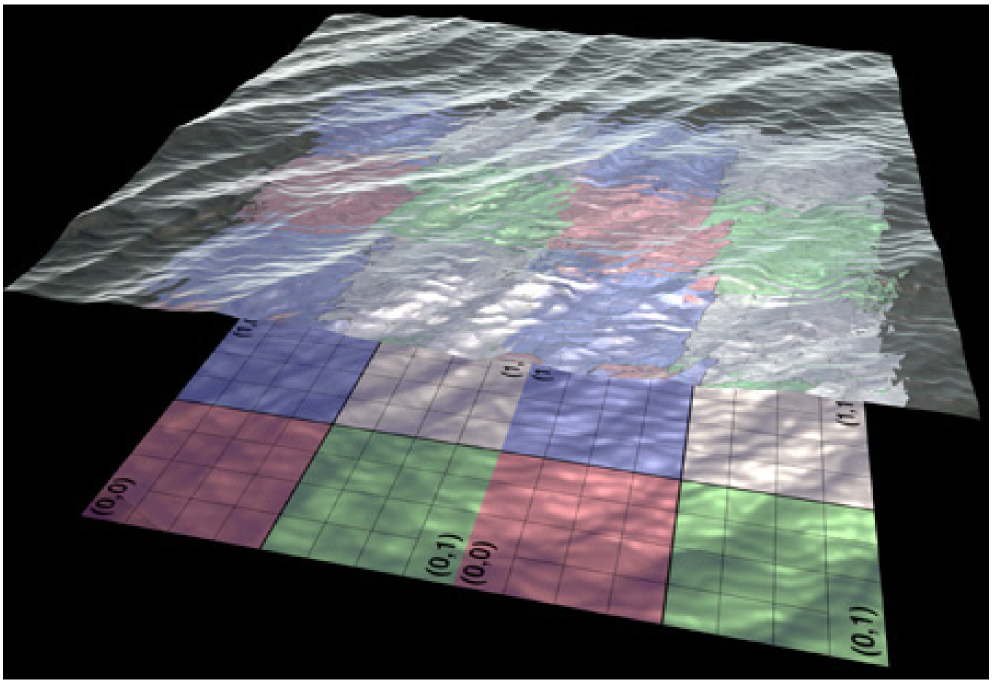
(Rendering time: 7 seconds)

● **Note**

In order for your displacements to be visible in reflections and/or refractions, you will need to add the *Trace Displacements* attributes to your *shape*. First, select the geometry, then, in the Attribute Editor, go to the Attributes menu and select *RenderMan > Manage Attributes*, select *Trace Displacements* from the left-hand column of the Add/Remove Attributes window, and click on the *Add* button.

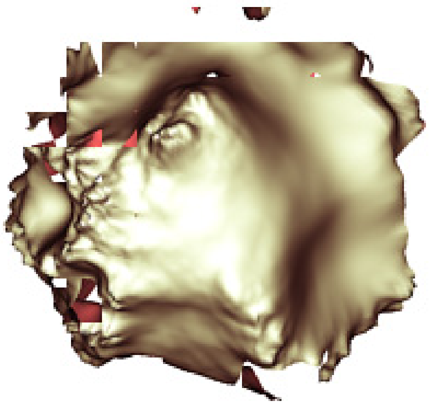
3 SUMMARY

Displacements can be used to create detail efficiently in a scene without adding modeling detail. By pushing fine details into a displacement shader (using either procedural functions or texture maps), 3D objects can be modeled with less detail, which can be a big benefit when working with large datasets and complicated geometry. For instance, in the image below an otherwise flat plane has been made more interesting by adding an animated wave displacement (and ray traced refractions):

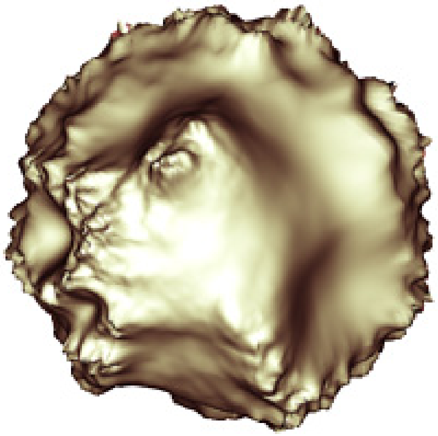


Displaced waves and ray-traced refractions

While RenderMan displacements are both detailed and fast, there are a couple of issues that you should be familiar with. The most important concept is *displacement bounds*. Displacement bounds set up a bounding box around the object, for use when the object is rendered. The bounding box determines when the object is loaded by RenderMan. If a displacement shader pushes an object outside of its bounding box, you will see that part of the displacement is being clipped, like in this image:



Incorrect Displacement Bound



Correct Displacement Bound

The solution in this case is to increase the bounding box. If you need to adjust displacement bounds, simply add the RenderMan displacement attributes to the shader. You can do this by following these steps:

```
1) Open the displacement shader in the Attribute Editor.
2) From the Attribute Editor Menu:
        Attributes-> RenderMan-> Add Displacement Attrs
```

Now that you've added displacement attributes they will appear at the bottom of the shader under the *Extra RenderMan Attributes* tab. Open this tab and adjust the *Displacement Bounds* attribute. The correct setting will vary depending on the size of your object in world space. Generally, a good value to start with is the farthest distance an object may be displaced, as measured in default Maya units, and adjust from there. Note that too large of a displacement bound can cause an object to consume more memory than needed, so the tightest displacement bound possible is recommended.

For more information on displacements see: **Bump and Displacement Shaders**.

Prev | Next

Particles

To get started, let's open the Maya scene, *particles.ma*.
(Where are the tutorial files?)

1 SOFTWARE RENDERING

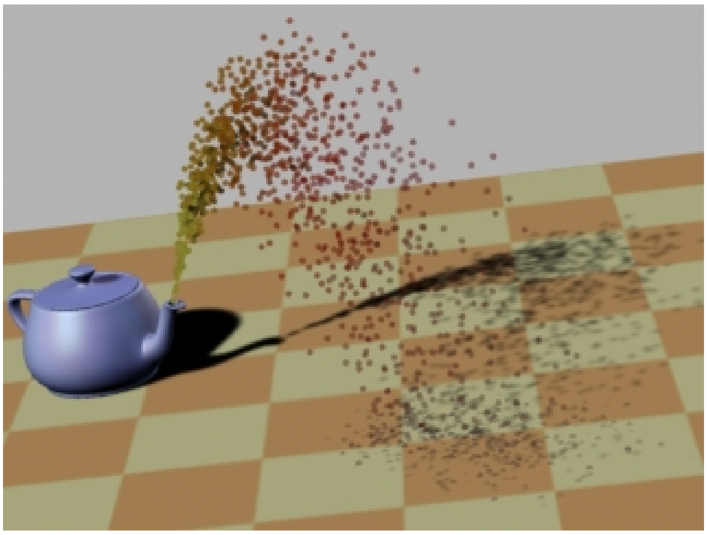
Open the scene, *particles.ma*, in Maya. For this tutorial we have a teapot spouting particles into the air. In this tutorial we'll cover the basics of using particles with RenderMan for Maya. This tutorial assumes you are familiar with creating particles and Maya Materials using Maya.

The first thing you'll notice is that RenderMan for Maya renders particles in software rendering. By rendering in software the particles will be correctly anti-aliased, motion blurred, self-shadowing, and fully integrated into the final rendered image.

Render the Maya scene with RenderMan.

```
Render-> Render Current Frame
```

You should get an image like the one below. Notice how Pixar's Deep Shadows add rich, semi-transparent shadows to the particles:



Particle points rendered in sofware

RenderMan for Maya supports nearly all of the Maya particles. Here are the supported particle types:

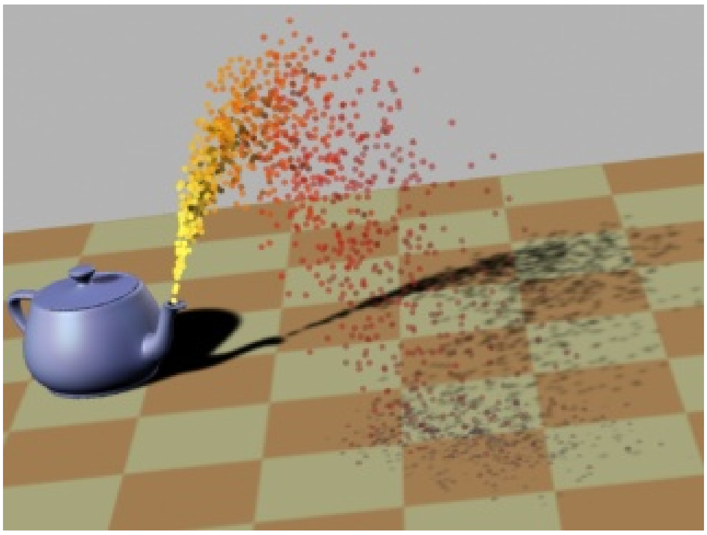
MultiPoint	Supported
MultiStreak	Supported
Points	Supported
Spheres	Supported
Sprites	Supported
Streak	Supported
Blobby Surface	Supported
Cloud	Rendered as blobby surfaces
Tube	Partial support
Particle Cloud Shader	Minimal support

Next we'll take a look at what can be done with several of the particle types in RenderMan for Maya.

2 POINTS: TWEAKING THE PARTICLES

Because we're rendering particles in software mode, we can tweak the Lambert material attached to these particles. We can make the particles brighter by selecting the Lambert and changing its color from grey to white. Note you can adjust any of the Lambert parameters: Diffuse, Transparency, etc. For now just change the color from grey to white, and render the Maya scene:

You should get an image with brighter particles.



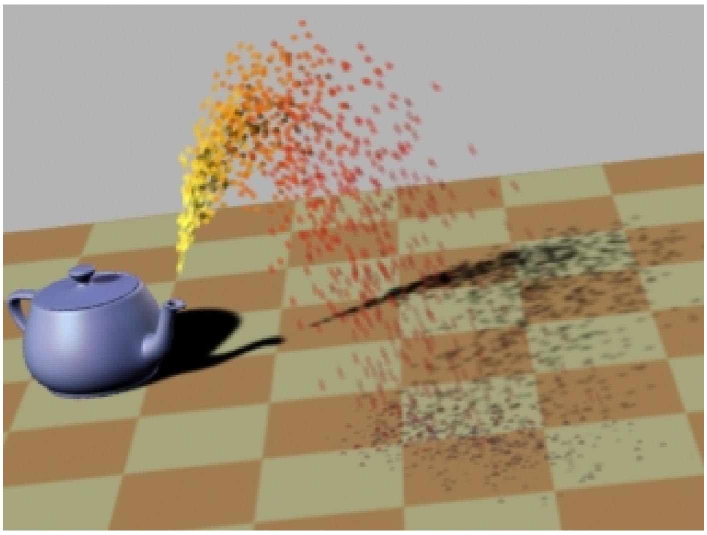
Brighter particles by changing Lambert's color to white

3 POINTS: ENABLE MOTION BLUR

Motion blur would be a nice effect to add to these particles. We can add motion blur by opening the Render Globals and selecting the *Features* tab. Enable motion blur by clicking the appropriate check box (next to *Motion Blur*). Now Render the Maya scene:

```
        Render-> Render Current Frame
```

You should get an image like the one below. One of the nice things about Deep Shadows is that the shadows are motion blurred, too.

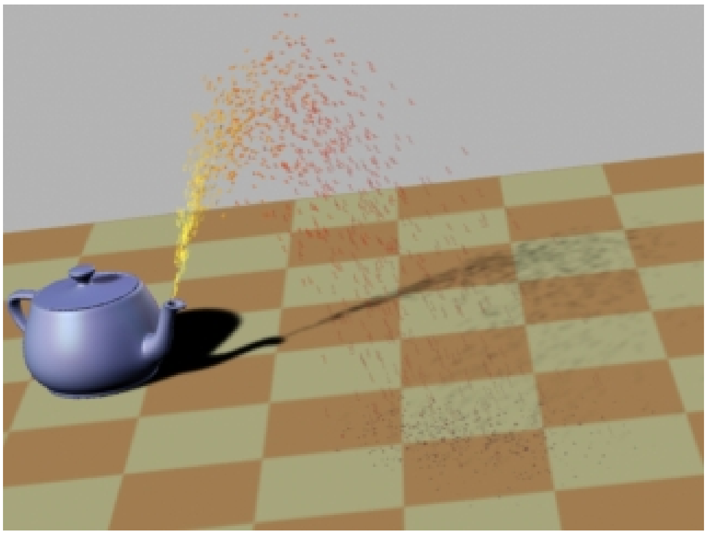


Motion blur enabled

4 POINTS: PARTICLE ATTRIBUTES

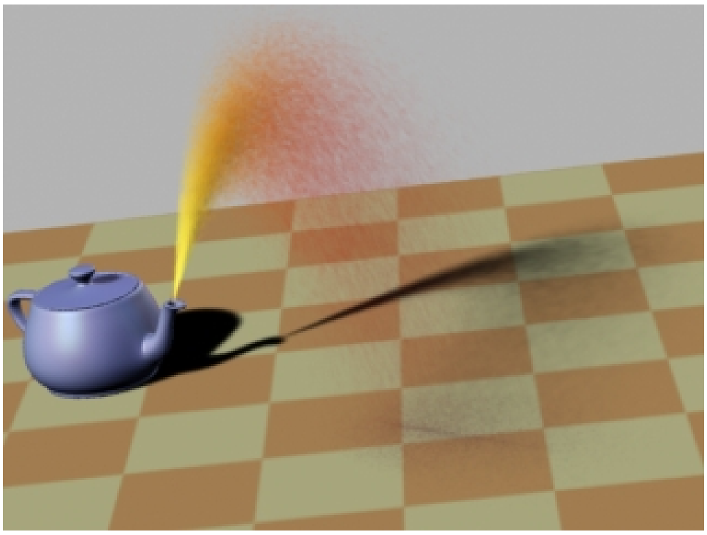
You'll also notice that RenderMan for Maya honors Maya's per-particle attributes, like color and opacity. This particle system already has per-particle attributes for color and opacity set for you.

Next add the particle attributes for this specific partical system, points. Do this by selecting the "Current Render Type" button next to the *Add Attributes* field of the *Render Attributes* sub-tab. Now we adjust additional attributes for the points particle type. You'll notice that RenderMan renders points in world space, not pixel space, which has the advantage of giving the points an actual size in the Maya scene. We can make the points smaller by setting the *Point Size* parameter to "1." Now render the Maya scene, you should have smaller points:



Smaller points

Because RenderMan can handle huge amounts of points, a common technique is to set the points to be nearly transparent and render a much larger amount. So next: change the rate of particles per second from 300 to 30000, set the particles to nearly transparent in the Lambert material, and do the particle run-up again. Now render.

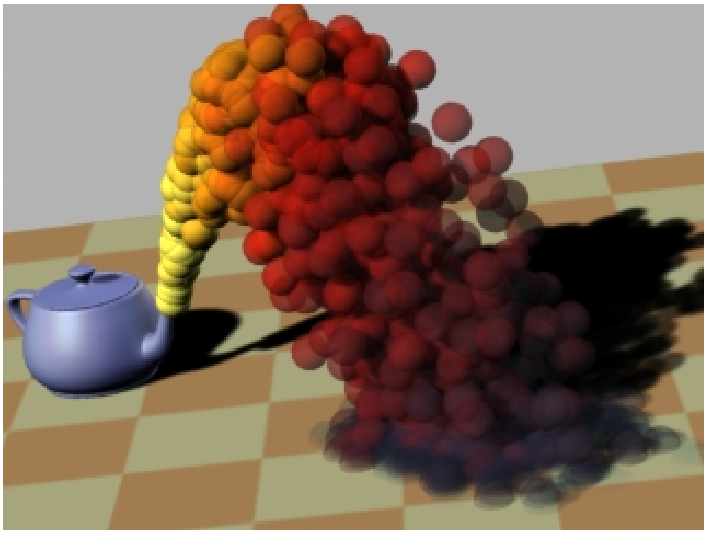


120,275 particle points

5 SPHERES: POINTS TO SPHERES

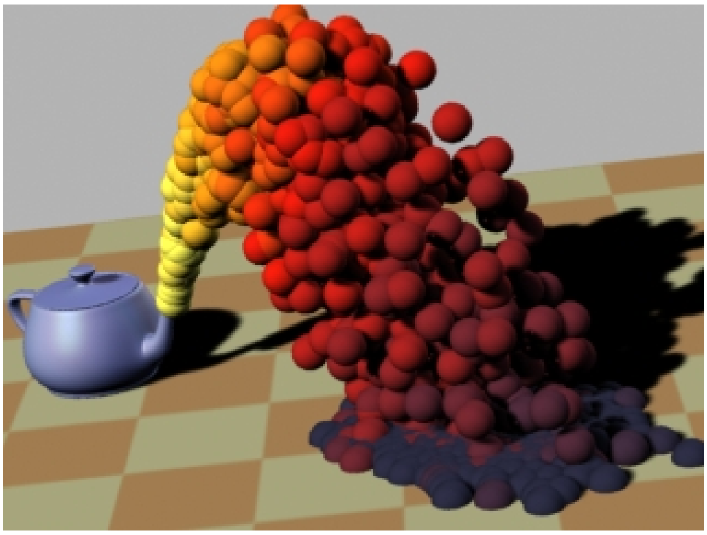
Next we'll work with the Maya sphere particles. Close the scene and then reopen, *particles.ma,* to restore the default settings for the scene.

Now select the particles and change them from points to spheres. Now render the Maya scene; you should get spheres.



Self-shadowing semi-transparent spheres
(Rendering time: 114 seconds)

You'll notice that the image took a long time to render. The reason the image takes long to render is the same reason the shadows (particularly the self-shadowing) look good ... the Deep Shadows are storing a lot of transparency data and that gets more expensive the more transparent objects there are. In this case, because there are so many transparent objects, the scene is quite expensive. To speed it up we can get rid of the transparency, so select the particles and delete the per-particle opacity attribute. Now render again ... notice that it renders much faster:



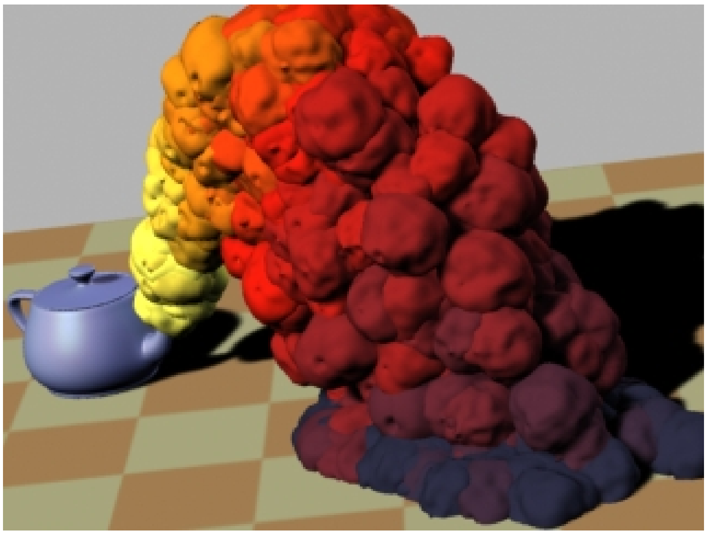
Sphere particles without transparency
(Rendering time: 43 seconds)

It's good to remember that Deep Shadows can create special effects like shadow transparency, but Deep Shadows can also be expensive to render.

6 SPHERES: ADDING DISPLACEMENTS

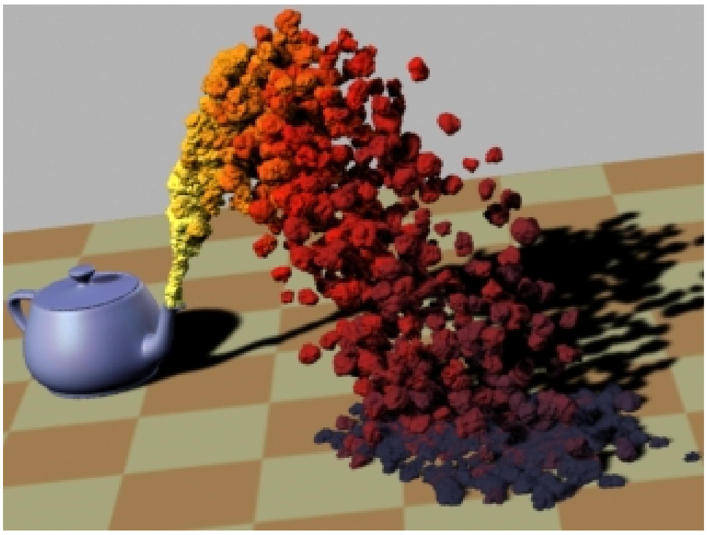
With RenderMan, we can break up the regularity of the spheres by adding displacements to them.

To do this, select the Lambert shader attached to the spheres and connect a Solid Fractal as a displacement (from the Lambert's Shading Group node). Now render:



Spheres with a displacement shader

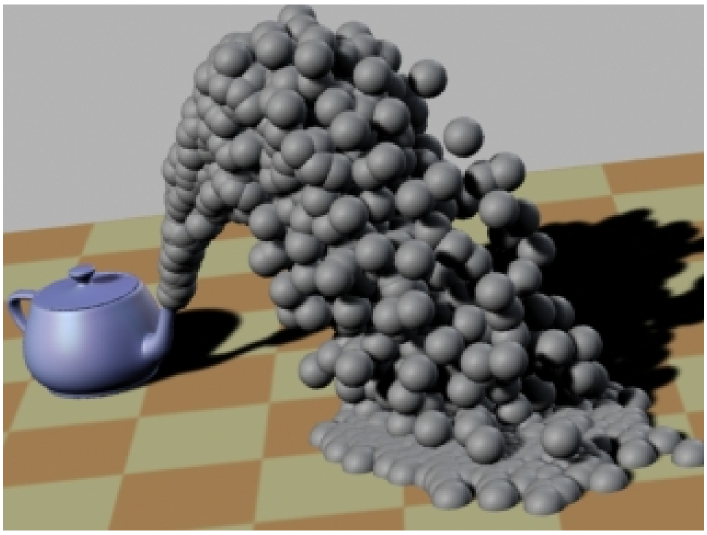
We can now tweak the parameters of the displacement to create a wide variety of effects. For now we'll just set the *Alpha Gain* of the displacement's *Solid Fractal* to "-0.5" to push the displacement back in toward the particles. Now render:



Displacement with an Alpha Gain of "-0.5"

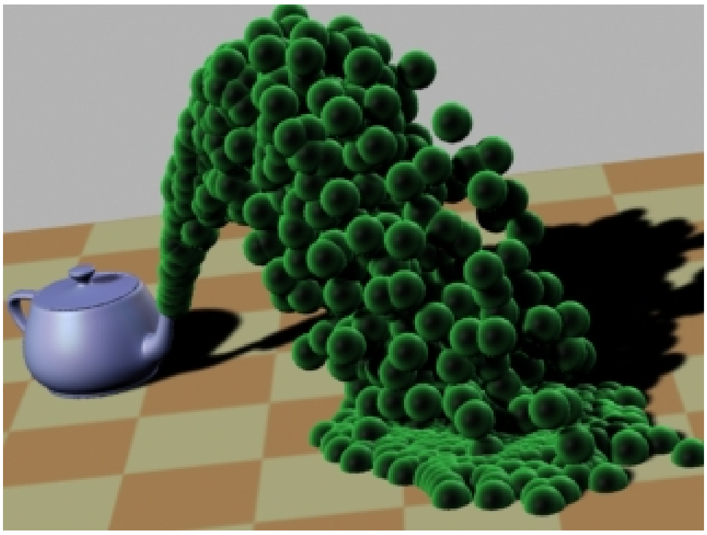
7 SPHERES: ADD ALTERNATE MATERIALS

Additionally, other Maya Materials can be attached to particle systems. Lambert reads per-particle attributes and should be used in those cases where per-particle attributes are required, but, alternatively, other Materials can be attached to particle systems to create certain kinds of effects (while ignoring per-particle attributes, like color and opacity). To demonstrate this, select the particle system and attach a Maya Ramp Shader. Now render:



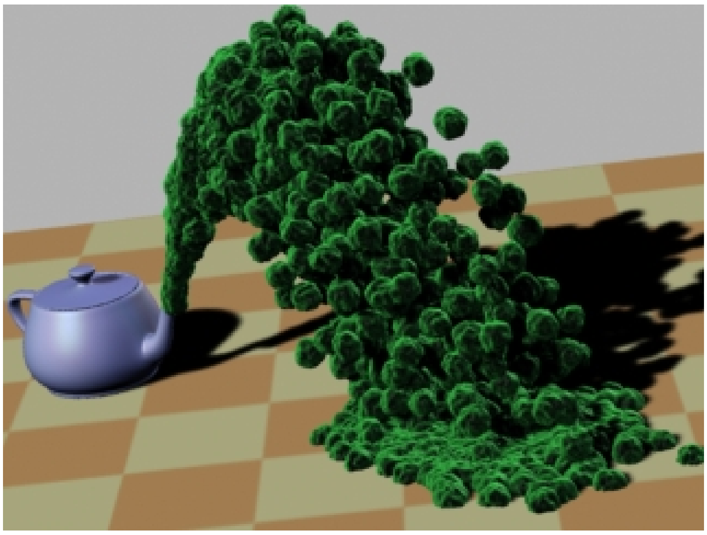
Spheres with a Ramp Shader

The spheres are using the Ramp Shader, which means we can create interesting effects, like changing the Ramp Shader's *Color Input* to *Facing Angle* and ramping the color from black to green. Go ahead and do just that (the black color should face the camera and the green should face toward the sides). Now Render:



Spheres with Facing Angle

Now we can add another displacement shader. Connect a Solid Fractal to the Ramp Shader, as we did above. Set the *Alpha Gain* of the displacement's *Solid Fractal* to "-0.2" to push the displacement back in towards the particles. Now render:



Displacement with an Alpha Gain of "-0.2"

5 SUMMARY

RenderMan for Maya is capable of rendering nearly every Maya particle type and provides advanced features for making particles look as good as possible. Some of the advantages of rendering particles in RenderMan are:

1) Particles are rendered in software ... providing correct anti-aliasing, motion blur, and complete integration.
2) Rich complex shadows (supporting semi-transparency, motion blur, and self-shadowing) are easy to create using Pixar's Deep Shadows.
3) RenderMan particle points are highly efficient.
4) Displacement shaders can be added to sphere particles.

Prev | Next

Fur and Hair

To get started, let's open the Maya scene, *fur_hair.ma*.
(Where are the tutorial files?)

1 WITHOUT FUR

Render the Maya scene, *fur_hair.ma*, with RenderMan.

```
Render-> Render Current Frame
```

You should get an image like the one below:

A bald teapot

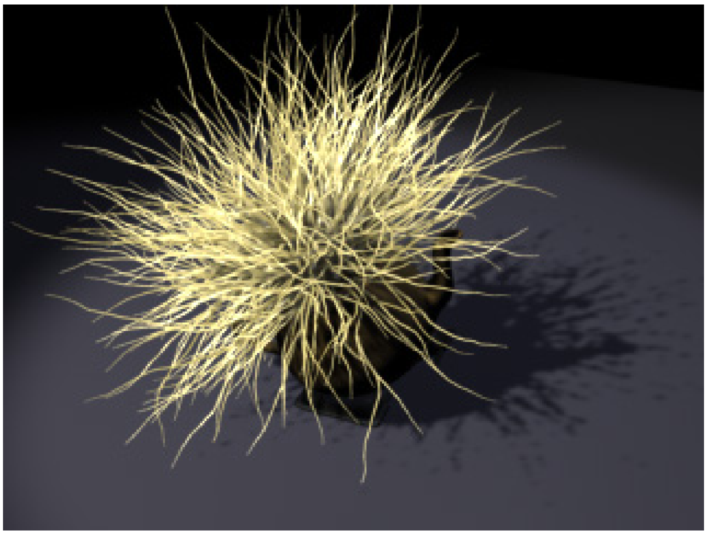
2 ADDING FUR

For this scene we'll use Maya's default *Porcupine* fur (found on the Maya shelf named, *Fur)*. Attach the Porcupine fur to the part of the teapots named *Scalp*, which can be easily found in Maya's Outliner. The scalp geometry is the large part flat part of the teapot's lid.

When you've attached the fur, simply render again.

```
Render-> Render Current Frame
```

You should get an image with fur:



Teatpot with a head of hair

You may notice three things. First, setting up fur is easy (and RenderMan For Maya supports animated fur, too). Second, the fur is rendered in software ⟩ as little pieces of curved geometry (it isn't a post-process); by rendering fur in software mode we gain the benefit of this tight integration and, most importantly, we get proper anti-aliasing. Third, you'll notice that the shadow bites . The shadow is chunky and pixelated ⟩ this is a problem inherent in depth map shadows. Fortunately, RenderMan has something called *Deep Shadows* which have what it takes to get fur to cast nice-looking shadows. How? We'll attach one next.

3 ADDING A DEEP SHADOW

When rendering fur, Pixar's unique shadow map format, Deep Shadows, comes in handy. Deep Shadows are a special type of shadow map that can store transparency information and can also be filtered. In fact, Deep Shadows can even be motion blurred ⟩ these are things that you just cannot do with a traditional depth map shadow, and these are also things that are essential to creating acceptable shadows for fur.

●Tip

> The reason why we don't use Deep Shadows all the time is that they are more expensive than normal shadow maps. Unless you need the features of Deep Shadows, it's generally best to use regular shadow maps.

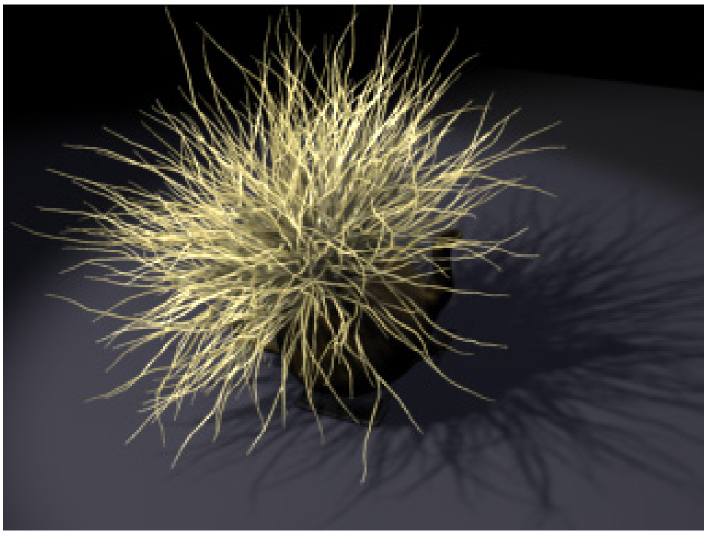
To create a Deep Shadow instead of a normal shadow map, simply add the RenderMan shadow attributes to the light that is casting the shadow. In this case the light is called *key*. To attach the attributes, follow these steps:

```
1) Select the light (in this case named "key").
2) Open the light in the Attribute Editor.
3) From the Attribute Editor Menu:
        Attributes-> RenderMan-> Add ShadowMap Attrs
```

Now that you've added the shadow map attributes, they will appear at the bottom of the light shader under the *Extra RenderMan Attributes* tab. The shadow map attributes can be set to create Deep Shadows or normal depth maps. Since the attributes default to Deep Shadow mode, you can simply render the scene again:

```
Render-> Render Current Frame
```

You should get an image like the one below:



Fur with deep shadows

You can see that the fur with Deep Shadows is free of any chunky artifacts. This is especially important when animating fur. Chunky artifacts become very obvious. Using fur and Deep Shadows together is highly recommended indeed.

4 ADDING MAYA HAIR

The workflow for Maya Hair is exactly the same as it is for fur: create Maya hair, animate, tweak hair, and render. Remember, hair looks best with deep shadows, too.

- **Important**

  RenderMan for Maya only provides complete support for Maya Hair beginning with Maya 6.5.

5 SUMMARY

RenderMan for Maya will render both Maya Fur and Maya Hair, including key-frame animation and dynamics. Since fur is rendered in software (and is not a post-process) RenderMan will create fur and hair that is properly motion blurred and anti-aliased.

For more information about fur see: **Fur**.

RenderMan Features

You can follow along with tutorials by using the Maya files included with the documentation. The tutorial files are collected in a Maya Project and can be located here: **doc_location/Recipes_and_Tutorials/rfm_project** (right-click to save). Save the folder called ***rfm_project*** into your Maya Project directory, update Maya's project location, and you'll be ready to go.

Global Illumination

To get started, let's open the Maya scene, *global_illumination.ma*.
(Where are the tutorial files?)

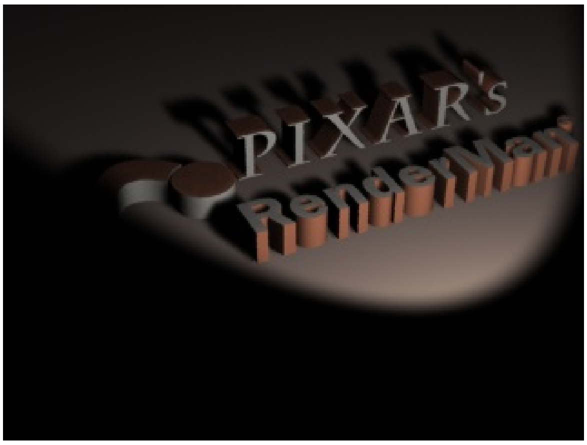
1 BEFORE GLOBAL ILLUMINATION

RenderMan for Maya lets you create global illumination effects in a variety of ways. Global illumination is a term that is widely used to refer to effects that create subtle soft shadows in a scene. To create global illumination effects we'll use the RenderMan Environment Light, which has several schemes for creating these types of effects. As we use these different schemes, you'll begin to understand how to most effectively employ the RenderMan Environment Light.

First, let's render the Maya scene, *global_illumination.ma*.

```
Render-> Render Current Frame
```

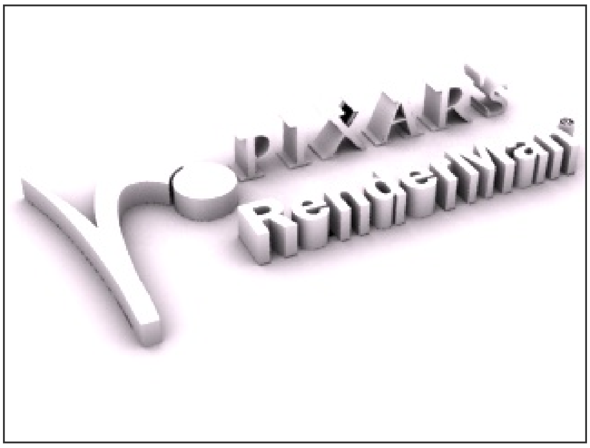
You'll get an image like the one below:



Direct lighting. No global illumination effects.

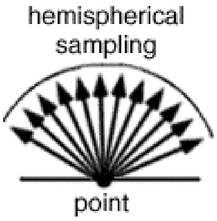
2 ADDING OCCLUSION

Perhaps the most production-ready scheme for creating global illumination effects is occlusion. Occlusion is an effect that simply determines how much a surface point is obscured by other surfaces. This technique can be used to create highly realistic images. The best means to motivate occlusion is probably to show a picture, so here's a picture of the scene above rendered only with the occlusion effect (no other shaders or scene lights were used):



Pure occlusion

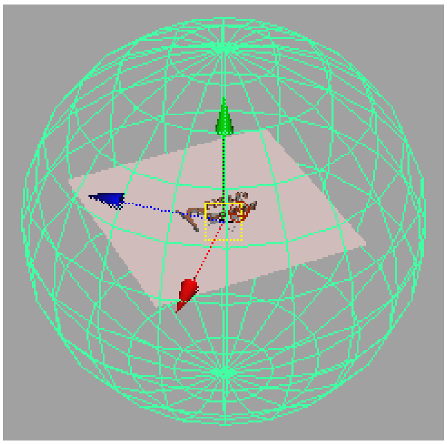
Occlusion creates nice diffuse shadowing. Keep in mind that this is an expensive ray-traced effect, requiring any given point to send many (sometimes thousands) of rays into a scene. Sending many rays from a point like this is called hemispherical sampling. The end result of hemispherical sampling is the average amount that a point is covered. Here is a diagram:



Now we'll create this occlusion effect. To add occlusion we'll create a RenderMan Environment Light:

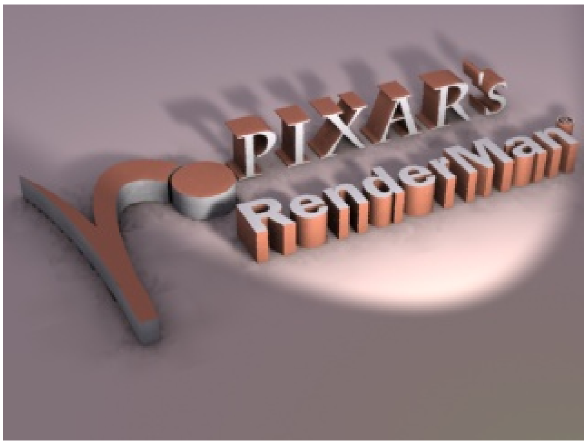
```
1) Open the Render Globals.
2) Select the "Features" tab.
3) Enable the "Ray Tracing" check box.
4) Under the "Global Illumation" create an "Environment Light."
```

By following the instructions above, a RenderMan Environment Light will be added to your scene. In the image below we can see that a RenderMan Enviroment Light has been added to the Maya scene.



The RenderMan Environment Light

Adding a RenderMan Environment Light is all you need to do to enable default occlusion. The color of the occlusion effect is determined by the environment color of the RenderMan Environment Light. Now render again ... occlusion is an expensive effect, but after a while you should get an image like the one below:



Default occlusion

The occlusion effect contributes to the existing lighting in a scene. You can see the subtle shadowing that occlusion adds. You'll also notice some artifacts in the occlusion, and that's because occlusion defaults to faster, lower quality, settings. Next we'll explore these quality/speed controls.

3 TWEAKING OCCLUSION

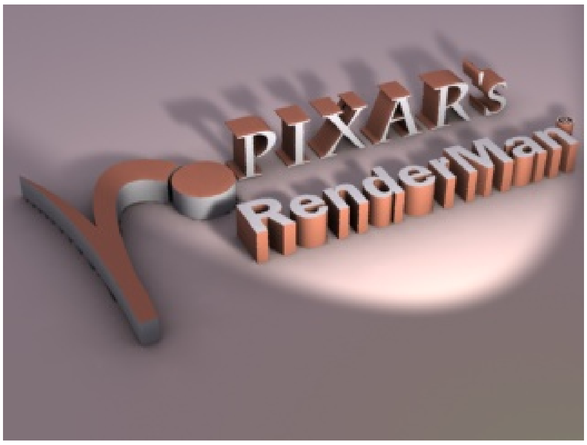
When rendering occlusion effects, preview renders can be accelerated by using quality settings that are faster, yet lower quality, because some artifacts may be acceptable in order to get faster iterative renders. On the other hand, a final rendered sequence requires high-quality, artifact-free settings. There are two main quality controls for occlusion: *samples* and *max variation*.

> **Samples** Controls how many hemispherical samples a point on a surface casts into a scene. More samples means higher quality. 64 samples is low quality while 1024 samples is high.
>
> **Max variation** Controls how often points on surfaces perform hemispherical sampling. 10 is low quality, while 1 (or lower) is high quality. Next we'll increase the quality settings for this image. A Max Variation setting of 0.0 uses no approximation, and is the highest quality setting. However, a setting of 1.0 is usually sufficient for production work.

To increase the quality settings, open the *RenderMan Enviroment Light* in Maya's Attribute Editor:

```
1) Change the Sampling Mode to "Sampled".
2) Set Samples to 256.
3) Set Max Variation to 1.
```

Setting the *samples* to 256 casts more ray samples, creating smoother shadows. (A higher quality sample rate would be 512 or 1024.) Setting *Max Variation* to 1 causes less approximation to occur, also increasing the quality of the shadows. Now render again:



Occlusion with higher quality settings

With higher quality settings the occlusion effect is much smoother, but it also takes longer to render. It's a quality/speed trade-off.

You'll find the RenderMan Environment Light also provides a number of parameters for adjusting the artistic nature of the effect. The occlusion intensity can be changed, an environoment map can be added, etc. You'll want to be sure to familiarize yourself with these features.

4 REUSING OCCLUSION

Because occlusion is an expensive effect, RenderMan for Maya allows you to reuse these calculations by caching the data on disk.

Currently, the occlusion effect is being calculated when the final image is rendered. To cache the data, we'll bake the occlusion in a pre-pass (similar to how a shadow map is calculated during a pre-pass and automatically reused in the final image). To bake occlusion do the following:

1) Select the *RenderMan Environment Light*.
2) In the Attribute Editor, right-click in the *Bake* parameter's field and select *rmanMakeGlobalDiffuse3dPass*.
3) The rmanMakeGlobalDiffuse3dPass tab will open. Leave the settings at their defaults.

In order to reuse the data we must first bake it. We've just set up RenderMan for Maya to bake the data. So we can render again:

```
Render-> Render Current Frame
```

When the image is rendered, the occlusion will be baked in a pre-pass and stored to disk. Next we'll reuse this baked data. To reuse the data do the following:

1) Open the bake pass in the Attribute Editor by clicking on the connection button next to the Environment Light's *Bake* parameter.
2) With the *rmanMakeGlobalDiffuse3dPass* open, switch the *Caching Behavior* from *Compute* to *Reuse*.

When RenderMan for Maya renders the scene it will now reuse the occlusion data, resulting in a much faster rendering. Voila!

Reused occlusion

Tip

You can bake occlusion into a 3D texture format (called a Brick Map), or you can bake the data into a 2D texture (camera projection). The right choice will depend on your particular purposes; you'll want to experiment with both. The 3D format is generally a good (camera independent) choice.

You can also configure the occlusion pass directly through *Passes* tab in the Render Globals. The *Passes* tab gives you a high degree of control over the individual passes that are part of any render job. By selecting a different pass in the Pass Settings window, parameters will automatically appear for that pass. Now that we've successfully reused occlusion, we'll delete the pass because it is not needed for the rest of the tutorial. To delete the pass, do the following:

1) Open the *Render Globals*.
2) Select the RenderMan *Passes* tab.
3) You'll see a list under *Pass Settings*, select *rmanMakeGlobalDiffuse3dPass* so it is highlighted.
4) Press the *Delete Pass* button to remove the occlusion pass.

5 ADDING COLOR BLEEDING

(Before going on to this step make sure you've deleted the occlusion pass as shown above.)

Color bleeding is an expensive ray tracing effect, more expensive than occlusion, but it has the additional feature of allowing colors to bleed from one surface onto another, making it useful in those instances when that added realism is required. Next we'll create this effect. To switch from occlusion to color bleeding, do the following:

1) Open the RenderMan Environment Light in the Attribute Editor.
2) In the "Shadowing" parameter, select "Color Bleeding".

That's all you need to do to enable color bleeding. To speed up the rendering a little more, we'll lower the quality settings. Return to the *RenderMan Environment Light*:

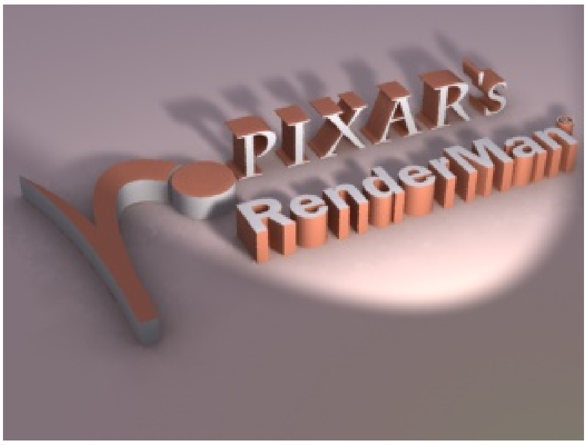
1) Set Samples to 64.

2) Set Max Variation to 5.

Tip

The artistic look of *color bleeding* can be dramatically changed by adjusting Max Dist. The appropriate setting for a scene will be determined by the distances between objects and the artistic goal. By lengthing and shortening the distance rays are shot, the objects that contribute to the color bleeding effect can be controlled.

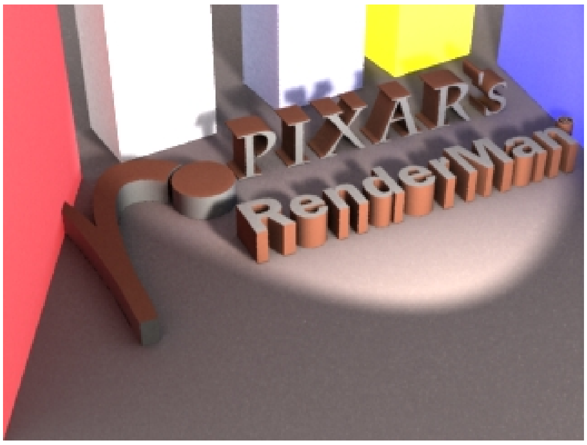
Now render again. This image may take a couple minutes to render. After a while you should get an image like the one below:



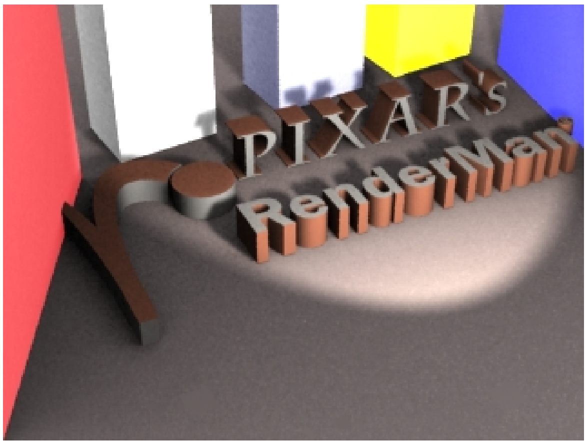
Color Bleeding

You'll notice this image looks very similar to the occlusion effect. However, if you look closely, you'll notice some color bleeding (between the floor and the base of the RenderMan lettering, for instance).

The difference between occlusion and color bleeding is more apparent in scenes that have materials with bright colors. In this example, there is a layer named "color_bleeder" with several colorful objects. This layer is currently invisible. Make the layer "color_bleeder" visible and render the scene again with color bleeding. The effect of color bleeding should now be more visible.



Rendered with color bleeding



Rendered with occlusion

Before we go to the next step, be sure to hide the *color_bleeder* layer. It is no longer needed.

Note

There are four special parameters for controlling the strength of global illumination effects at the shading module level:

- `Colorbleeding Scale` a multiplier for incoming intra-object colorbleeding.
- `Radiosity Scale` a multiplier for outgoing diffuse energy (colorbleeding).
- `Diffuse Environ Scale` a multiplier for incoming indirect diffuse environment light.
- `Specular Environ Scale` a multiplier for incoming specular environment light.

These can be added to most materials by selecting *RenderMan-> Add Environment Controls* from the Attributes menu of the Attribute Editor.

6 IMAGE BASED LIGHTING

So far we've used occlusion and color bleeding to create subtle shading effects in this scene. In both cases, however, the color of the environment light has been white. Instead of using a simple white environment color, we can add an environment map, a technique that can be used to simulate complicated lighting quickly. The technique of using an environment map to illuminate a scene is called *image based lighting*.

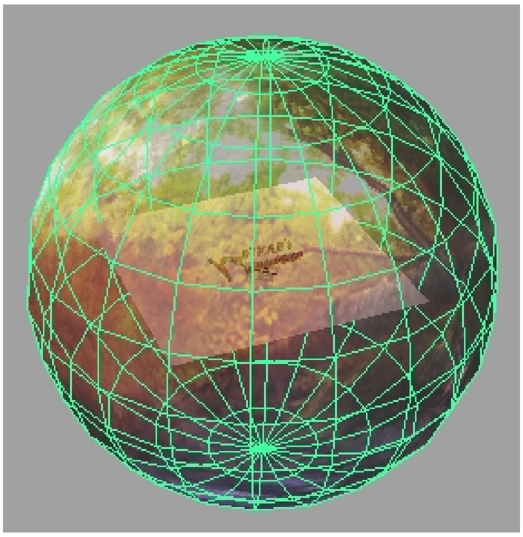
Image based lighting can use many kinds of image formats. RenderMan for Maya supports both HDRI and normal images. Here's the environment map we'll use in this scene. It's a normal image, not an HDRI ...

ibi.tif

In this example, we'll load up an environment map. Here's how to load the environment map:

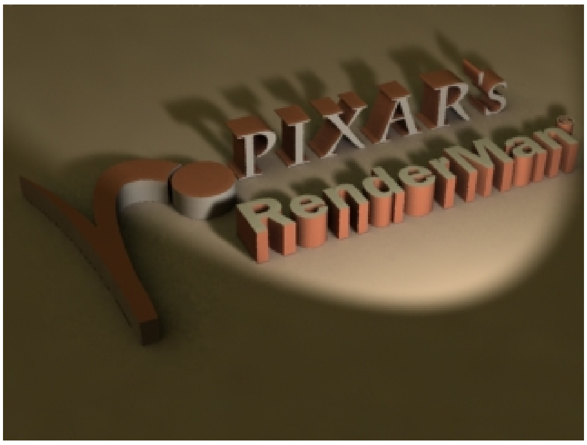
1) Open the RenderMan Environment Light in the Attribute Editor.
2) In the *Environment Map* parameter, browse for an image.
3) In the image browser select "ibi.tif" from this project's *sourceimages* folder.
4) Change the environment light's *Sampling Mode* from *filtered* to *sampled*.

Notice the RenderMan Environment light will display the image in Maya's interactive mode like this ...



Map displayed on the RenderMan Environment Light

Because occlusion is much cheaper that color bleeding, switch the *Shadowing* in the RenderMan Environment light to *Occlusion*. Now render again. The environment map will now cast light into the scene, like this:



Occlusion with an environment map

Notice how image based lighting techniques have been combined in the image above. To create pure image based lighting, delete the spot light. Now all of the illumination in the scene will come from image based lighting. To brighten the effect of the environment map, increase the *intensity* of the RenderMan Environment Light to "4". Now render again. The environment map will now illuminate the scene by itself:

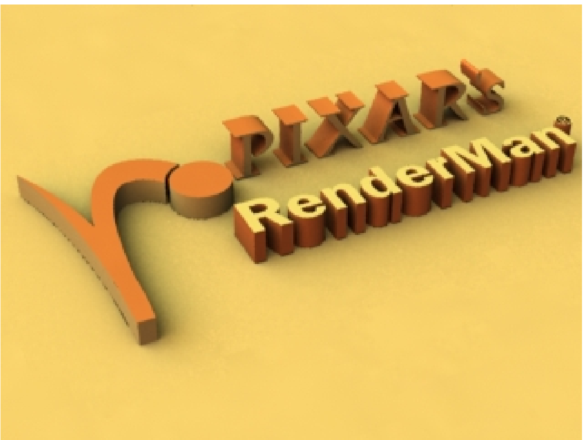


Image based lighting with occlusion.

●Tip

The combination of an environment map and occlusion can create some interesting effects, but you'll notice that the shadows created with occlusion are a function of geometry and not the image map. For more accurate shadows (especially when using HDRI), an enviroment map should be combined with color bleeding, and while this technique is quite expensive to compute, it is the most accurate image based lighting method available (if you can afford to render it).

---

7 POINT-BASED GLOBAL ILLUMINATION

The basic workflow for using point-based global illumination (approximate occlusion and color bleeding without ray tracing) is fundamentally the same as what we've described above. There is one major difference in the approach, however, which requires one simple step and introduces a new pass to your render. Point-based global illumination depends on a pre-baked point cloud, so you need to create a baking pass via the Bake parameter of the Environment Light node. You can choose either a single-pass approach, `RenderRadiosity`, or a two-pass approach for reusing the data, `MakeApproxGlobalDiffuse`. Right-click in the Bake field and select your method of choice from the list presented.

The Pass window is essentially the same as any other, but we want to pay particular attention to a couple of parameters that might otherwise go unnoticed. Point-based GI produces excellent results at coarser shading rates, so the first thing we want to note is that the default shading rate is 100. This is far from production quality, but you will find that you don't need to crank it down as far as you'd think in order to get excellent results. Additionally, ray tracing is explicitly disabled for this pass (though it need not be).

Take some time to run through the steps outlined above using point-based instead of ray-traced global illumination. You will probably need to tweak a parameter or two to optimize the quality of your results, but it won't take long for you to see the benefits of the new technique.

8 SUMMARY

You are able to create occlusion and color bleeding effects with RenderMan for Maya by creating a RenderMan Environment Light in the Render Globals. These effects can be baked during a pre-pass which is referenced when the final image is rendered, just like shadow maps. Occlusion is a production-ready effect. Color bleeding is more accurate but carries a greater overhead than occlusion. Additionally, by adding an environment map to the RenderMan Environment Light, image based lighting techniques can be employed.

Prev | Next

Subsurface Scattering

To get started, let's open the Maya scene, *subsurface_scatter.ma*.
(Where are the tutorial files?)

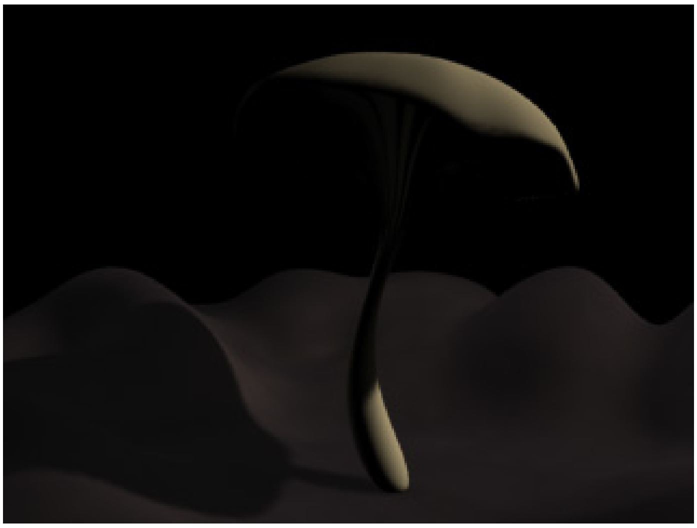
1 BEFORE SUBSURFACE SCATTERING

RenderMan For Maya lets you create subsurface scattering effects by attaching special RenderMan attributes to Maya Materials. When these attributes are attached to a Material a subsurface scattering pre-pass will be generated automatically, much like creating a shadow map in a pre-pass.

Before we get started let's render the Maya scene, *subsurface_scatter.ma*, without subsurface scattering.

```
Render-> Render Current Frame
```

You'll get an image like the one below:



The basic scene

2 ADDING SUBSURFACE SCATTERING

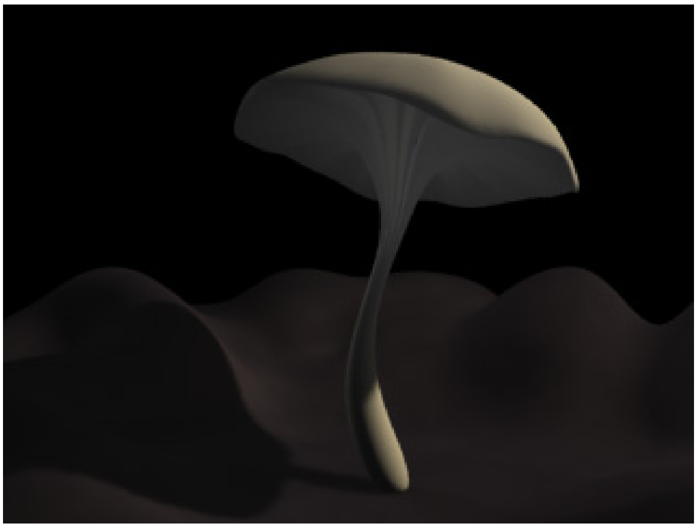
To add subsurface scattering, select the Maya Material called *Mushroom1* in the Hypershade. Next, we'll add the subsurface scattering attributes to this Material. To attach the attributes follow these steps:

```
1) Select the Material in the Hypershade (in this case named "mushroom1").
2) Open the Material in the Attribute Editor.
3) From the Attribute Editor Menu:
        Attributes-> RenderMan-> Add Subsurface Scattering
```

Now that you've added the subsurface scattering attributes, you will find them at the bottom of the Material under the *Extra RenderMan Attributes* tab. There are a number of subsurface scattering parameters that you can use to control the effect. Right now we'll render with the defaults, so simply render again:

```
Render-> Render Current Frame
```

You'll see the pre-pass generated first. After that is processed (in about 30 seconds), you should get an image like the one below:



Default subsurface scattering

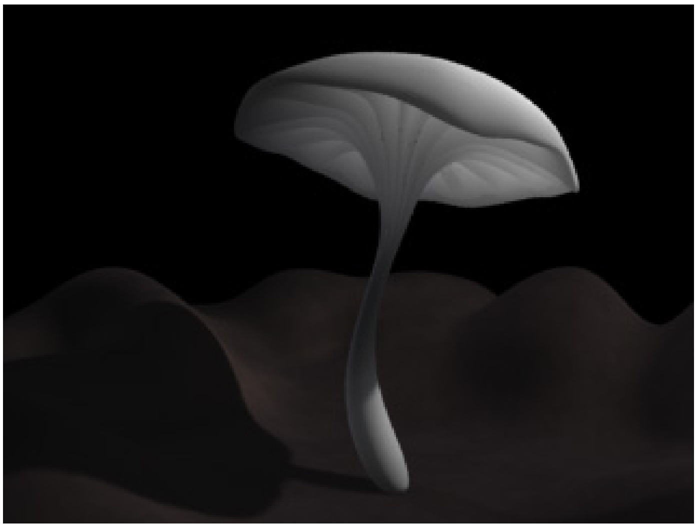
You can see the effect is there, but it needs to be tweaked for this particular model. We'll do that next.

3 TWEAKING SUBSURFACE SCATTERING

In the *Extra RenderMan Attributes* section of the Attribute Editor you'll find the parameters for controlling the subsurface scattering effect. The most important setting is the *Scattering Free Path* parameter, which determines how far light penetrates into an object. In this case we'll set the Scattering Free Path to 2 (as measured in world space). Render again:

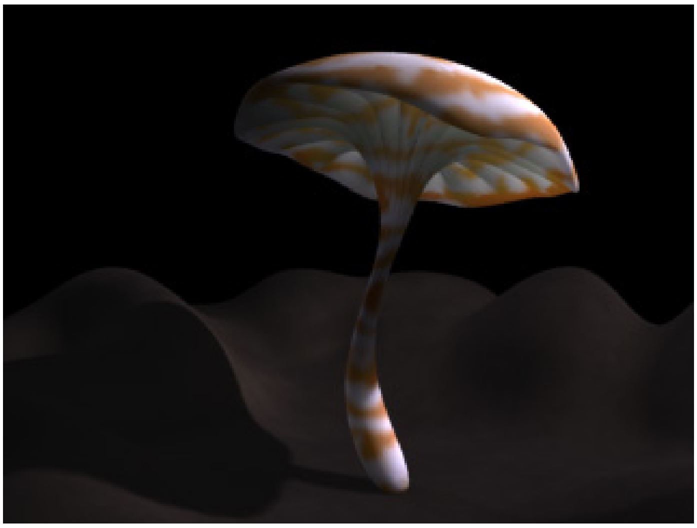
```
Render-> Render Current Frame
```

You should get an image like this:



Scattering Free Path set to "2"

By messing around with the other parameters (for instance, connecting a Marble to the *Albedo*), you can create some interesting effects, like this:



With Marble added to the Albedo

4 REUSING SUBSURFACE SCATTERING

Because subsurface scattering is a potentially expensive effect, RenderMan for Maya allows you to reuse these calculations. If the light doesn't change and the object isn't moving, reusing these calculations can save a lot of time. To reuse subsurface scattering do the following:

```
1) Open the Render Globals.
2) Select the RenderMan Passes tab.
3) You'll see a list under Pass Settings, select rmanSSMakeBrickmapPass.
4) Switch the Caching Behavior from Compute to Reuse.
```

The Passes tab gives you a high degree of control over the individual passes that are part of a render job. By selecting a different pass in the Pass Settings window, parameters will automatically appear for that pass. In this case, you've switched the mode of the Subsurface Scattering pass from *Compute* to *Reuse*. Now that you've set the Caching Behavior to Reuse, try rendering again and you won't have to wait for the pre-pass.

●Tip

Remember you'll want to switch the *Caching Mode* from *Reuse* to *Compute* if the lights change, the object moves, or if you're tweaking most of the subsurface scattering parameters.

5 SUMMARY

You are able to create subsurface scattering effects with RenderMan for Maya by attaching special RenderMan attributes to Maya Materials. This effect is generated in a pre-pass which is referenced in during the rendering of the final image, just like shadow maps.

●Tip

If you are rendering subsurface scattering on multiple objects, like a clump of mushrooms (where each object is clearly distinct from, yet close to, the others) you'll want to apply a separate material to each object. If a single material is used the subsurface scattering effect with be blurred across the objects.

Prev | Next

Deep Shadows

To get started, let's open the Maya scene, *deep_dragon.ma*.
(Where are the tutorial files?)

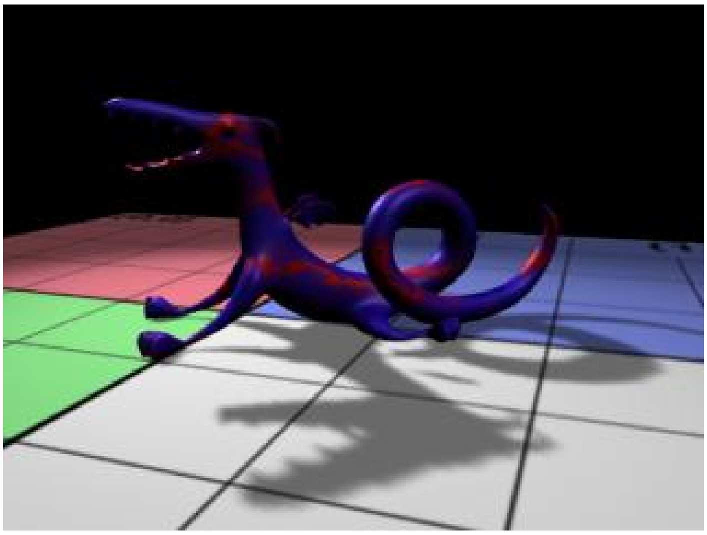
1 BEFORE DEEP SHADOWS

RenderMan For Maya supports Pixar's Deep Shadows. A Deep Shadow is a kind of depth map shadow that offers higher-quality shadows and supports transparency, motion blur, multi-sampling, and more. For more details, check out the Shadows section of the *Features and Effects* documentation.

Before we get into the finer points of using Deep Shadows we should have a frame of reference. Render the scene as is, using standard depth map shadows.

```
Render-> Render Current Frame
```

You should get an image something like this:



Depth Map Shadows

First we're going to modify the material attached to our plane to give us a good reference point. Select the `GroundPlane` material (it's just a Lambert) and let's map a simple image of a grid to the color. The image is located in the sourceimages folder of the `rfm_project` (`~/rfm_project/sourceimages/grid.tif`).

2 ADDING DEEP SHADOWS

Deep Shadows are enabled by adding a special RenderMan attribute to your light. In this particular case, we are adding it to the `Key` light (a spot light). Select the Key light and add a RenderMan ShadowMap attribute via the Maya Attribute Editor:

```
Attributes-> RenderMan-> Add ShadowMap Attrs
```

Scroll to the bottom of the Attribute Editor window and you will see that an **Extra RenderMan Attributes** section has been added. Click on the arrow to expand the section and see the Shadow Map attributes, including Pass Class, Bias, Samples, Filter Size, and Filter. The defaults here will be fine for our purposes.

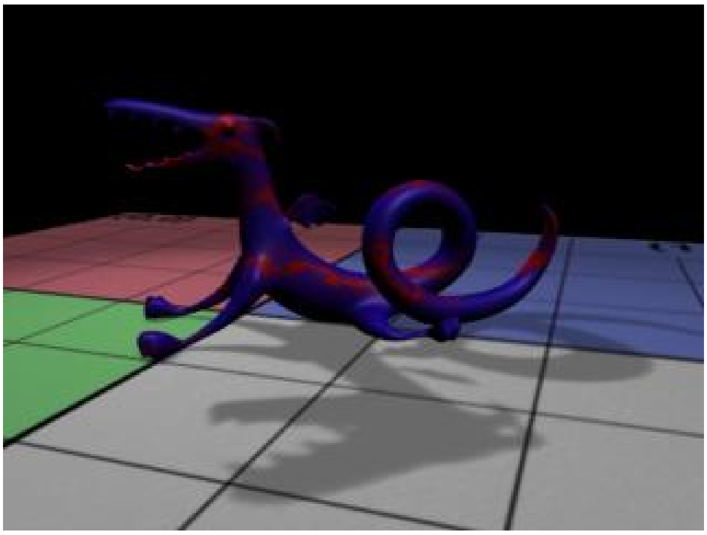
● **Note**

In adding the attribute you have also created a separate Deep Shadow pass for the renderer. Open the Render Globals, click on the *Passes* tab and select *DeepShadow Globals* to see the default settings for the pass, which we will discuss in detail later.

Let's see what happens now ...

```
Render-> Render Current Frame
```

You should see something not at all unlike this:



Pixar's Deep Shadows

You'll notice that the shadow is ... well ... prettier, more defined, but still plenty soft.

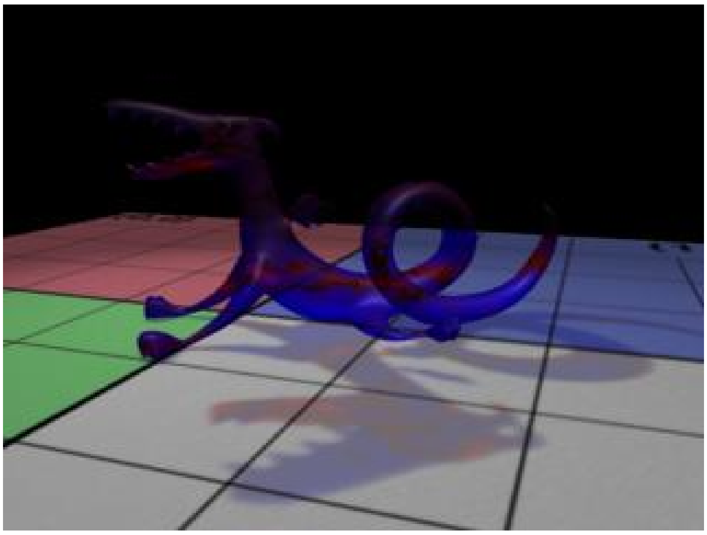
3 TRANSPARENCY!

Among other things, Deep Shadows do a fine job of giving you transparent shadows. These are oh so very useful for things like fur, particles, or perhaps even see-through dragons. Why don't we take a quick look ...

We can achieve this nifty effect quite simply, by mapping our Material's color to our Material's transparency. Our dragon has a Marble node attached to the color, so we'll want to add a connection to the transparency as well. Using Hypershade, you could connect the `outColor` of the Marble material (`marble1.outColor`) to the `transparency` of the dragonMaterial (`dragonMaterial.transparency`), or you could middle-mouse drag the Marble from the Hypershade window and drop it on the Transparency for the dragonMaterial in the Attribute Editor, or you could› Well, there are a lot of ways to do it. Once you've done it, re-render.

```
Render-> Render Current Frame
```

And Presto-Change-o, your image should look like this:



A See-Through Dragon with Deep Shadows

If, for some reason, see-though dragons aren't your bag, take a look at the Fur and Particles bits of the Getting Started section to see just how spiffy deep shadows can be.

4 DIGGING DEEPER

As we were going through these steps you might have noticed a lot of parameters and drop-down menus and sliders and stuff that we just ignored, leaving them at their defaults. Here's a quick run-down of those bits we temporarily glossed over ...

Extra RenderMan Attributes

`Shadow Type`

This drop-down list defines the type of shadow that RenderMan will use. Its default value is DeepShadow.

`Shadow Bias`

This offsets the shadow map to prevent self-shadowing artifacts. The default, you might have noticed, is `0.010`.

`Shadow Samples`

Controls the number of samples used for the map. The higher the number, the smoother the shadow, and the greater the price in rendering time. The default is 16.

`Shadow Filter Size`

The size (in pixels) of the filter applied to the shadow map. Larger filters cause more blur.

`Shadow Filter`

You can toggle between gaussian or box filter types to choose the type of smoothing that you want applied to your shadow map.

DeepShadow Globals

`Phase`

Choose the frequency that the pass (deep shadow map generation, in this case) is run. Deep shadow maps, like any map, can be reused. If your map includes only non-moving objects, change the phase from *Every Frame* to *Once Per Job* to reuse the map.

`Shading Rate`

Your shadow map does not necessarily have to use the same shading rate as your final image. Set a rate for the shadow map here. It will have the same sort of speed/quality tradeoff associated with the shading rate for your final image.

`Pixel Samples`

The number of pixel samples in x and y for your shadow maps. When using deep shadows, you will get better results with pixel samples greater than one.

`Motion Blur`

You can disable motion blur or defer to the Render Globals' *Features* tab from this drop-down menu. You *must* enable motion blur via the *Features* tab if you want motion blur.

`Ray Tracing`

This has the same effect as the `Motion Blur` setting, disabling ray tracing or deferring to the Render Globals' *Features* tab.

`Default Surface Shader`, `Expand Surface Shaders`, `Expand Displacement Shaders`, `Expand Light Shaders`, `Expand Volume Shaders`

Deep shadows use information from these types of shaders, so the shaders must be expanded during the deep shadow pass. A standard shadow map, on the other hand, would only need information from the displacement shader; if you click on the *Shadow Pass* entry in the *Passes* tab you can see that only `Expand Displacement Shaders` is enabled by default.

Determines the size of your shadow map.

●Tip

RenderMan for Maya's Deep Shadows respect Maya's *Disk Based Dmaps* setting; by changing it to Reuse Existing Dmap[s] your deep shadows can be cached and reused, saving valuable render time.

---

5 SUMMARY

Deep shadows are an easy way to get high-quality effects which would normally require more expensive ray tracing to achieve, but they are still larger and slower than traditional shadow maps. Like any feature that carries an added expense, they should only be used when needed. Take some time to explore the possibilities of using deep shadows, such as their usefulness with fur and particles and their support for motion blur (for more on motion blurred deep shadows, check out the aforementioned Particles section of the documentation). Used properly, deep shadows can be a great help in generating high-quality imagery.

Prev | Next

RenderMan Shaders

Fot this tutorial open the Maya scene, *renderman_shader.ma*.
(Where are the tutorial files?)

1 THE MAYA SCENE

In this tutorial we will take a look at using the RenderMan Shader node. Please note that writing RenderMan Shaders is not a basic skill and is beyond the scope of this tutorial. However, there are numerous resources available for RenderMan Shaders, as well as GUI-based tools, such as Pixar's Slim, included with the RenderMan Artist Tools, that can enable you to tap into the robust RenderMan Shading Language.

The RenderMan Shader node references compiled RenderMan shaders (.slo files). RenderMan for Maya includes the shader utility, `shader`, which allows you to compile shader source (.sl) files. Alternatively, a tool like Slim can create compiled shaders for you.

Open up the Maya scene, *renderman_shader.ma*. This scene is simple: the proverbial teapot, in a box. The scene should look something like this:

Maya Scene

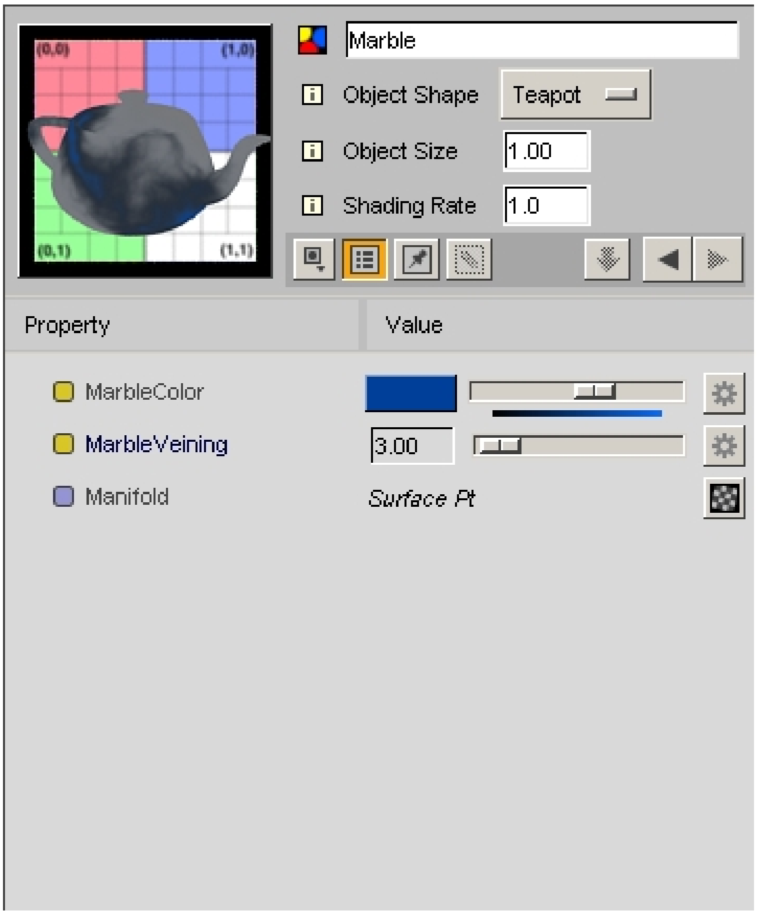
2 SHADER SOURCE

We're going to use both raw `.sl` files and compiled shaders built with Slim in this exercise. These can be found in shaderexamples directory of the rfm_project. The shader source files are borrowed from *The RenderMan Companion*, a classic blue marble surface shader and a similarly timeless displacement shader. You can view the shading language code by opening these files (`blue_marble.sl` and `dented.sl`) with your favorite text editor. Here's the source for the dented displacement, just because:

```
displacement
dented (
        float Km = 1.0)
{
        float size = 1.0,
              magnitude = 0.0,
                  i;
        point P2;
```

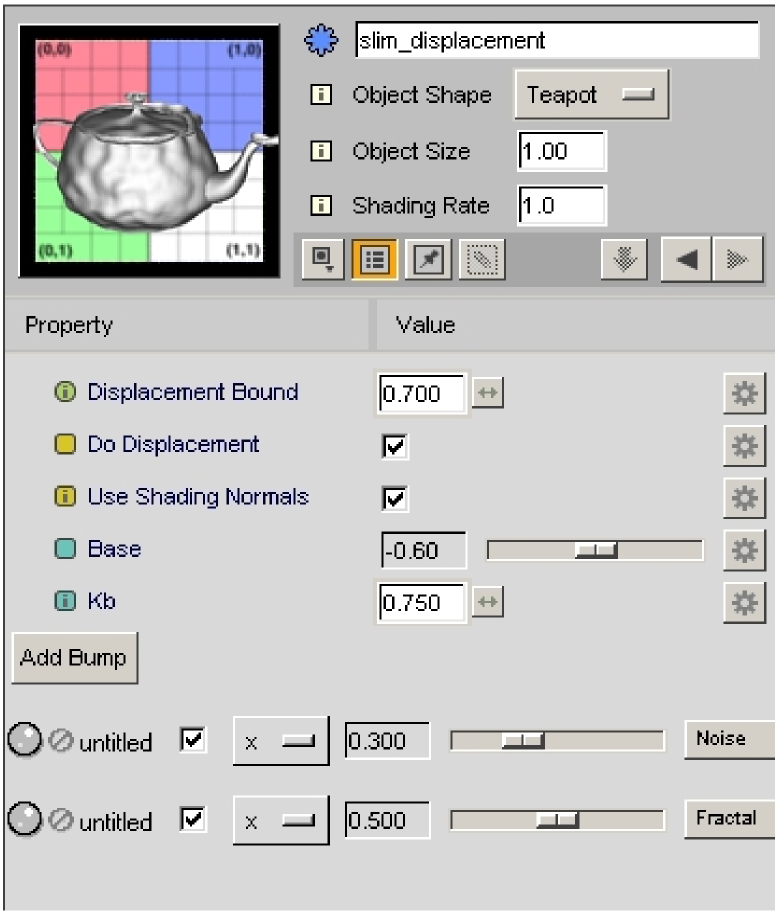
```
                P2 = transform ("shader", P);
                for (i = 0; i < 6.0; i += 1.0) {
                        magnitude += abs (.5 - noise (P2 * size)) / size;
                        size *= 2.0;
                }
                P2 = P - normalize (N) * (magnitude * magnitude * magnitude)* Km;
                N = calculatenormal(P2);
        }
```

We've created a couple of Slim appearances that are rough approximations of these shaders. There is a simple diffuse appearance created using the Delux node, and a Combine displacement that uses a noise and a fractal function in combination. If you have the RenderMan Artist Tools you can take a look at the palette, in the shaderexamples directory. Whether you have RAT or not, here's a look at the appearances:



Marble
Object Shape: Teapot
Object Size: 1.00
Shading Rate: 1.0

Property	Value
MarbleColor	
MarbleVeining	3.00
Manifold	*Surface Pt*



slim_displacement
Object Shape: Teapot
Object Size: 1.00
Shading Rate: 1.0

Property	Value
Displacement Bound	0.700
Do Displacement	☑
Use Shading Normals	☑
Base	-0.60
Kb	0.750

untitled ☑ x 0.300 Noise
untitled ☑ x 0.500 Fractal

Marble by Slim

Slim's Displacement

Note that the widgets of the parameters that were marked as external are a nice shade of blue-green/aqua/teal (e.g. *base* and *Kb*. in the displacement shader; these parameters will be exposed in the RenderMan Shader node in Maya and can be further tweaked.

3 COMPILING RENDERMAN SHADERS

The first thing that needs to be done to accommodate the RenderMan Shader node is: compile your shader. RenderMan for Maya includes the shader utility; run off the command line, shader compiles shaders. If you're using Slim, Slim does the compiling for you, each time you click on the preview swatch.

To use the shader utility you will need to set an RMANTREE environment variable (unless you also have RenderMan Pro Server installed, in which case you should have it set already). Otherwise the RMANTREE variable should point to the rmantree directory in your RenderMan for Maya or RenderMan Studio installation whichever you have installed.

So, on the Mac:

1. Open the Terminal utility (Applications>Utilities>Terminal).
2. Type csh to switch to the C Shell command line interpreter.
3. cd to the shaderexamples directory:
   `cd /path/to/rfm_project/shaderexamples`
4. Set the RMANTREE environment variable:
   `setenv RMANTREE /Applications/Pixar/RenderMan_for_Maya7.0-1.0/rmantree/`
5. Compile the shaders:
   `/Applications/Pixar/RenderMan_for_Maya7.0-1.0/rmantree/bin/shader blue_marble.sl`
   and
   `/Applications/Pixar/RenderMan_for_Maya7.0-1.0/rmantree/bin/shader dented.sl`

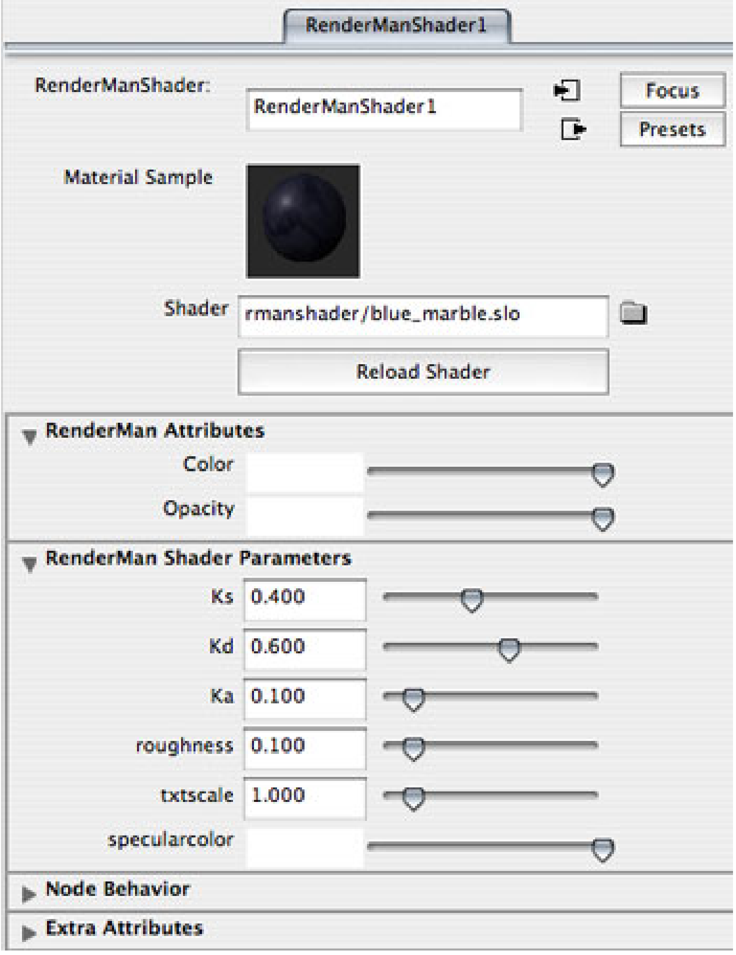
On Windows:

1. Open a Command Prompt window (Start>Programs>Accessories>Command Prompt).
2. cd to the shaderexamples directory:
   `cd C:\path\to\rfm_project\shaderexamples`
3. Set the RMANTREE environment variable:
   `set RMANTREE=C:\Program Files\Pixar\RenderManForMaya7.0-1.0\rmantree`
   - *NOTE*: If you use the tab key to auto-complete the directory names, the Command Prompt window will add quotation marks to the beginning and end of the path; these quotation marks must be removed.
4. Compile the shaders:
   `"C:\Program Files\Pixar\RenderManForMaya7.0-1.0\rmantree\bin\shader" blue_marble.sl`
   and
   `"C:\Program Files\Pixar\RenderManForMaya7.0-1.0\rmantree\bin\shader" dented.sl`
   - *ANOTHER NOTE*: Entirely unlike the preceding step, when you needed to make sure that you do not use the quotation marks, in this case you need to make sure that you *do* use the quotation marks. If you're not sure, you can copy and paste the commands above into the Command Prompt window.

If you look at the contents of the directory where the .sl files are, you will notice the presence of two new files, blue_marble.slo and dented.slo.

4 USING RENDERMAN SHADERS

As you might have guessed, given that we've been looking at a surface shader and a displacement shader, we're going to use two RenderMan Shader nodes in a single Shading Group. Go ahead and open Hypershade and click on the RenderMan Shader icon to create the first node. Click on the little folder icon next to the Shader field and browse to the compiled surface shader, blue_marble.slo. The Material Sample will update and the RenderMan Shader surface parameters will populate the Attribute Editor window. It will, in fact, look a little something like this:



RenderManShader: RenderManShader1

Shader rmanshader/blue_marble.slo

Color

Opacity

Ks 0.400

Kd 0.600

Ka 0.100

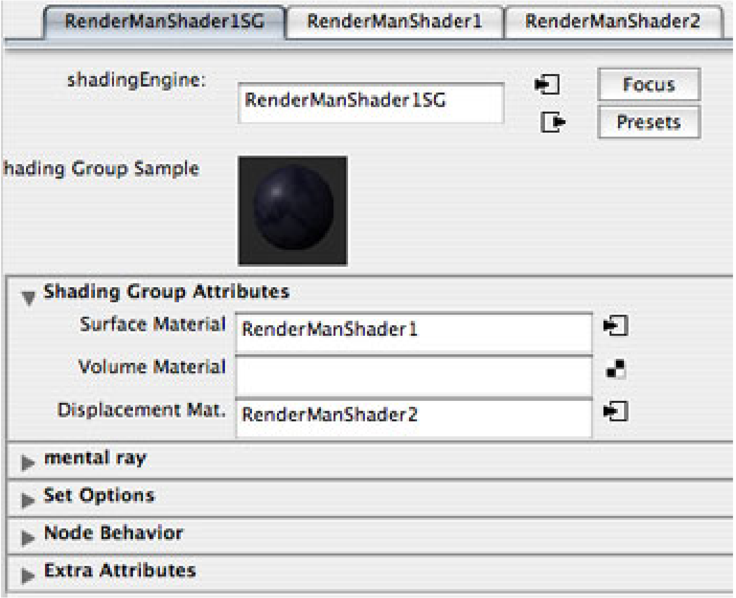
roughness 0.100

txtscale 1.000

specularcolor

The RenderMan Shader Node

Having accomplished this singularly spectacular bit, we could stop, but let's go ahead and add our displacement to the shading group. Go ahead and back up to the Shading Group level, `RenderManShader1SG`. You'll note that RenderManShader1 is referenced in the Surface Material field. Click on the Displacement Material attachment widget to connect our second RenderMan Shader. With Maya, at this point you would probably select a Texture in the Create Render Node window, but Maya considers the RenderMan Shader a Material, although it can be a surface or a displacement. Click on the *Materials* tab and select the RenderMan Shader node. This will open RenderManShader2 in the Attribute Editor; browse to dented.slo to attach your displacement shader. Your Shading Group will end up looking like this:



shadingEngine: RenderManShader1SG

Surface Material RenderManShader1

Volume Material

Displacement Mat. RenderManShader2

The RenderMan Shader Shading Group

Assign the Shading Group to the teapot and its lid, and *render!*

One Dented Marble Teapot

The same method applies for applying the shaders created with Slim. Open RenderManShader1 in the Attribute Editor again, click on the folder icon, and this time browse to the `slim_marble.slo` shader. What say we Tweak a parameter or two here? Note that the sliders adjust the values relative to the original values. Let's lower the `Diffuse_Intensity` a bit, say to 0.500, and increase the `Specular_Intensity` to 0.700-ish.

Now let's open RenderManShader2 and browse to the `slim_displacement.slo` shader. Let's bump the Kb up a bit, to 2.000 (Kb is a multiplier for the displacement), and let's lower the Base to -1.5. Render again.

A Subtly Different Dented Marble Teapot

● **Note**

You can make subsequent changes to the source for your RenderMan Shader node. If you edit your .sl file by hand or tweak your Slim appearance, simply recompile the shader and then click the *Reload Shader* button in the corresponding RenderMan Shader node.

5 SUMMARY

Integrating your RenderMan shaders with your Maya scene is just about as easy as pie using RenderMan for Maya's RenderMan Shader node. The node can be assigned just like any other Maya material, and is little more than a matter of browsing to your compiled shaders, which maintain a nice degree of tweakability even after being imported.

In addition to building shaders with Slim, there is a vast number of sources for shaders, both in print and on the Internet. Here are a few hefty tomes worth a look-see:

- *The RenderMan Companion: A Programmer's Guide to Realistic Computer Graphics*
  ISBN: 0201508680
- *Advanced RenderMan: Creating CGI for Motion Pictures*
  ISBN: 155860618
- *Rendering for Beginners: Image Synthesis Using Renderman*
  ISBN: 0240519353
- *Essential RenderMan fast*
  ISBN: 1852336080

Prev | Next

Secondary Outputs

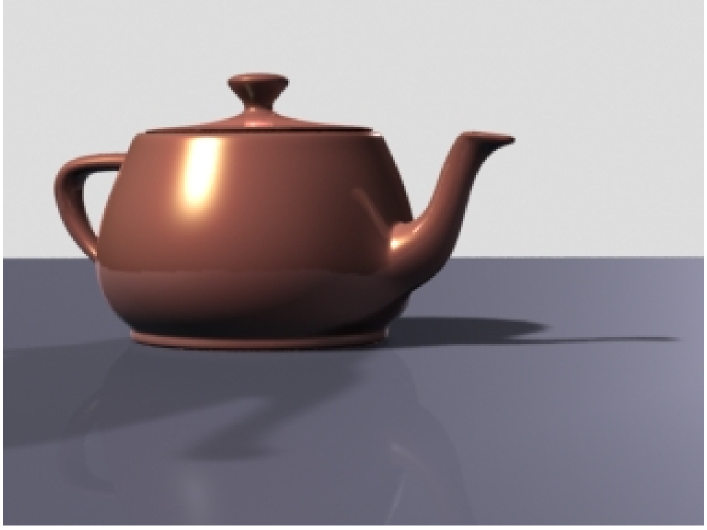
To get started, let's open the Maya scene, *outputs.ma*.
(Where are the tutorial files?)

1 SECONDARY OUTPUTS

Render the Maya scene, *outputs.ma*, with RenderMan.

```
Render-> Render Current Frame
```

You should get an image like the one below:



The final image

The rendering above is the final image, a combination of many different effects and surface qualities: diffuse coloring, specularity, reflections, etc. Sometimes it is advantageous to render these various qualities into their own image, as secondary passes. RenderMan allows you to render secondary passes at the same time the final image is rendered, allowing the final image to be output with a collection of secondary passes simultaneously. Secondary passes such as specularity, reflections, shadows, occlusion, etc., can be composited together after the fact, when specific elements of an image (like specularity) can be tweaked without having to render the whole image again. In this tutorial we'll go through the workflow of setting up secondary outputs with RenderMan for Maya.

2 ADDING A SECONDARY OUTPUT

First, we'll add a secondary output for the shadows, which will output the shadows into their own image file. To do this, follow these instructions:

1) Open the RenderMan Globals and select the *Passes* tab.
2) Locate the *Output* subtab of the *Passes* tab.
3) Open the pull-down menu (which looks like a downward arrow) next to the *Output* field and select:

```
Create Ouput-> DiffuseDirectShadow
```

That's it. You have now created a secondary output for a shadow. Whenever the final image is rendered, a secondary output for the shadow will be created at the same time. When this is accomplished, simply render again.

As before you should get the same final image:

The final image

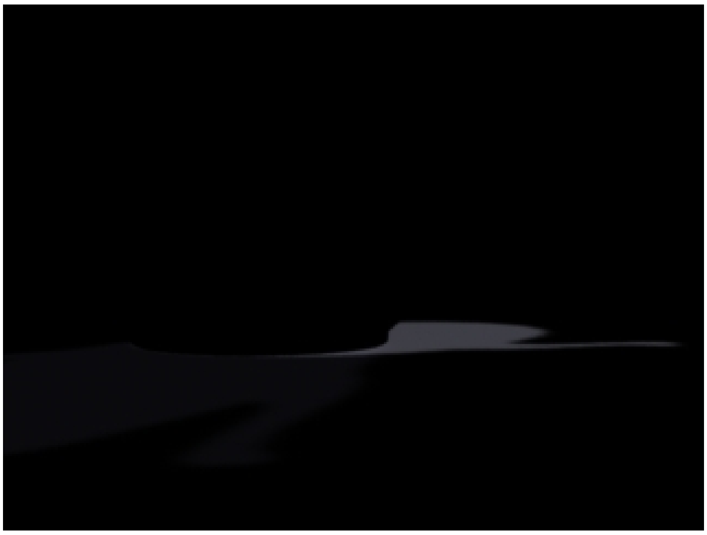
But now you'll also have the shadow output. This image is automatically stored in your current Maya project and is located in the *RenderMan* folder, under the name of the maya file. In this case the secondary output for the shadow can be found here:

```
RenderMan/outputs/images/outputs_DiffuseDirectShadow.iff
```

• Note

> RenderMan Studio users that are rendering to 'it' can see the secondary outputs via the Catalog view, with the images cascaded below each primary output.

The secondary output for the image looks like this:



The secondary output for the shadow.

3 ADDING MORE OUTPUTS

Using the same workflow, we can make secondary outputs for specular, reflections, and diffuse values. Do that now.

1) Open the RenderMan Globals and select the *Passes* tab.
2) Locate the *Output* subtab of the *Passes* tab.
3) Open the pull-down menu next to the *Output* field and select:
   `Create Ouput-> SpecularDirect`
4) Open the pull-down menu next to the *Output* field and select:
   `Create Ouput-> DiffuseDirect`
5) Open the pull-down menu next to the *Output* field and select:
   `Create Ouput-> SpecularIndirect`

Now render again. In addition to the final image and the shadow output, you'll also get the addtional secondary outputs in your directory, `RenderMan/outputs/images/`

The secondary output for the specularity.

The secondary output for the diffuse.

The secondary output for the reflection.

These individual layers are now ready for compositing.

4 MANAGING SECONDARY OUTPUTS

You'll notice that the pull-down menu next to the *Output* field has a list of the secondary outputs we just created. This pull-down menu can be used to toggle back and forth between secondary outputs. When a secondary output is selected, the parameters particular to that secondary output appear below it. Using these parameters any secondary output can be configured. The file format, the filter, the quantization, and more can be changed.

From this pull-down menu layers can also be deleted. Since we are done with this exercise, delete all of the secondary outputs that we just created:

1) Locate the *Output* subtab of the *Passes* tab in the Render Globals.
2) Open the pull-down menu next to the *Output* field and select:

```
Create Output-> Delete Output
```

●Tip

Usually secondary outputs should be created during the final render, but in certain situations you'll want to create secondary outputs during other pre-passes, like shadow passes, reference passes, etc. By switching pass types (using the *Pass* pull-down menu in the *Passes* tab) you can add secondary passes to any individual pass that you require.

5 SUMMARY

Secondary Ouputs, useful for compositing techniques, can be created with RenderMan for Maya. RenderMan for Maya is flexible when it comes to how these secondary outputs are created. Because the pixels for the secondary outputs are output at the same time that the pixels for the final image are calculated, RenderMan is very efficient when creating these outputs. In fact, secondary outputs require minimal additional overhead.

Prev | Next

Caustics

To get started, let's open the Maya scene, *caustic_dragon.ma*.
(Where are the tutorial files?)

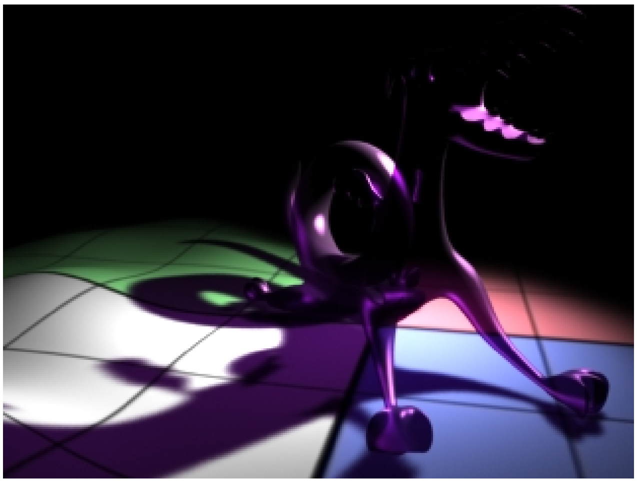
1 BEFORE CAUSTICS

RenderMan for Maya uses a classic two-pronged approach to rendering caustics, utilizing shading models and photon maps to create the effect. Caustics can be created using Maya materials or RenderMan shaders with equal ease and just a couple of tricks .

First, let's render the Maya scene as is.

```
Render-> Render Current Frame
```

You'll get an image like the one below:



Direct lighting, with transparent Deep Shadows.

2 ADDING CAUSTICS

The aforementioned two-pronged approach boils down to two things: surfaces and lights. The shading model is an attribute that you attach to your surfaces and the photon map is generated, using a special pass, from attributes added to your light.

As you can see from our scene, with the good ol' reliable dragon sitting on our oogly grid-plane-thing, we have two types of surfaces: one translucent surface filtering light to create the caustics and another opaque (and matte) surface that caustics are cast upon. We need to add RenderMan attributes to each to let the renderer know how to treat each during the caustics pass.

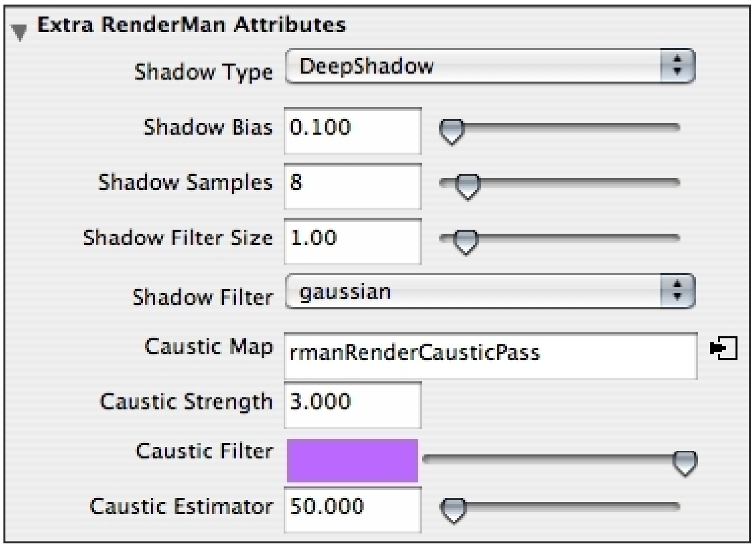
1. Select the *Dragonberry* material attached to our dragon (it's a Phong E, but a Blinn would do just as well) using your method of choice.

   From the *Attributes* menu in the Attribute Editor, select **RenderMan -> Add Caustic Controls**. In the *Extra RenderMan Attributes* section that is added there is now a drop-down list of Shading Models to choose from (note that it appears after the Refraction attributes that we added surreptitiously)); for our purposes, the Glass Shading Model will work best.

2. Select the *Gridbert* material and add the caustic controls to that as well. In this case, you want to select the Matte Shading Model.
3. Now select the *Key* light. Before we add the appropriate attributes, note that we've already added

RenderMan Shadow Attributes. Deep Shadows are not required for caustics, but they sure are nice. We'd be remiss if we did not point out that our Key light's falloff is set to *Quadratic*. This is not negotiable   you simple *must* do this)please. Because of the change in the falloff, we needed to compensate a bit by increasing the light's intensity. This light is, like, totally intense.

So)with that out of the way, go ahead and select **RenderMan -> Add Caustic Attrs** from the Attributes menu. They'll look a little something like this:



Extra RenderMan Attributes

Shadow Type: DeepShadow

Shadow Bias: 0.100

Shadow Samples: 8

Shadow Filter Size: 1.00

Shadow Filter: gaussian

Caustic Map: rmanRenderCausticPass

Caustic Strength: 3.000

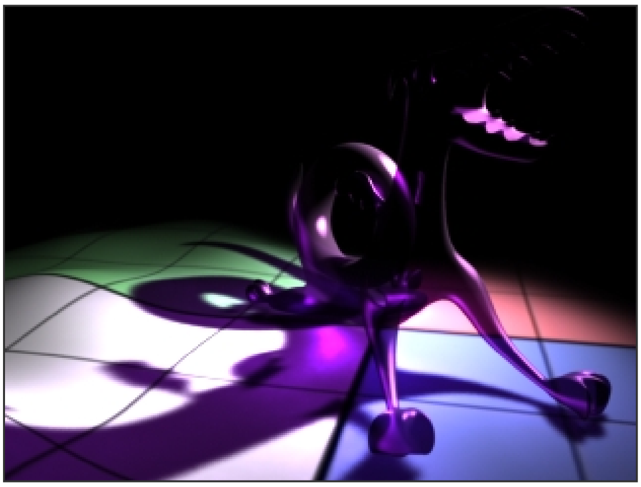
Caustic Filter: 🟪

Caustic Estimator: 50.000

The first thing you'll notice is that there's a *Caustic Map* parameter, and an `rmanRenderCausticPass` has been created. There are also parameters controlling the strength and coloring of the caustics, and a parameter (*Caustic Estimator*) that controls how many photons are consulted in the caustics calculations.

For our purposes, to make our a smidgen more overt and to pretty things up a bit, let's bump the Caustic Strength up to 3-ish, make our filter a lovely shade of lavender, and lower the Caustic Estimator to 50.

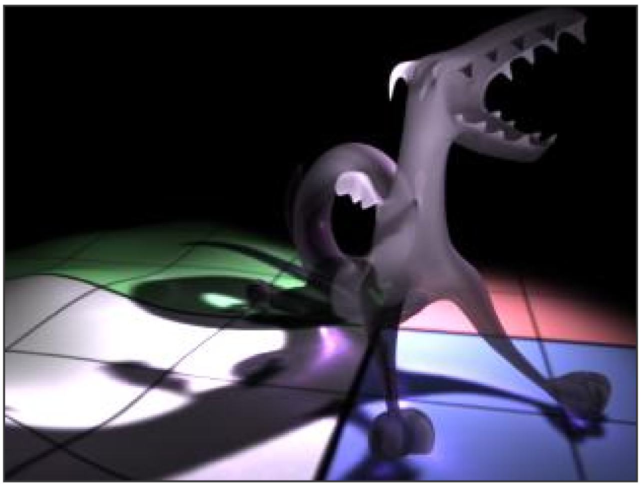
4. Now let's take a look at that pass we've created. Click on the widget next to the Caustic Map parameter to navigate to the pass tab (or you can access it from the *Passes* tab in the *Render Settings*). You've got your usual pass settings, for cameras, objects, lights, and caching, plust you have some caustic-specific settings. For now, the important ones are the *Emit* setting, which dictates the actual number of virtual photons to emit, and the Shading Rate. We want to bump the number of photons up to 500,000, and we'll leave the Shading Rate alone for now, but we want to remember it's there, for future reference.
5. That seems like a lot of stuff to do, but it went pretty smoothly, right? Okay then, let's render)



Voilà! A wee smudge of a bright spot in the middle of our shadow. Um)let's see if we can't spruce that up)go back to the caustic pass and spike the photons setting, then lower the shading rate. It will take a wee bit longer to render, but the results are nice   a bit more definition to the caustics and some nice spread, plus additional caustics under the front feet)

3 GOING PRO

You can get the same results with a RenderMan Shader using essentially the same workflow. RenderMan Studio users can build their appearance in Slim, add it to their scene, and then add the necessary attributes via the Attribute Editor, or anyone can create a RenderMan Shader node, reference an appropriate .slo file, and proceed as above. The image below is using a funky little Delux appearance created with Slim 7.0.



Caustics with a RenderMan Shader

4 CONCLUSION

Although there are a few important things to remember, creating caustic effects is an essentially simple process. After you're comfortable with the simple scene we've set up here, run through things again with a different caustic shading model, like *Chrome*. The workflow is the same, but you get the funky caustics of sunlight bouncing off your bling-bling.

Prev | Next

Pixar Animation Studios

RenderMan for Maya Pro Tutorials

Prev | Next

The RAT to RMS FAQ

A Transition Primer

Introduction

So, you're making the big move, transitioning from the RenderMan Artist Tools to the new and improved suite of rendering goodies, RenderMan Studio. If you are unfamiliar with the RenderMan for Maya workflow, you will find that there are a few tweaks to your workflow. This FAQ is designed to cover the basics of the adjustment, easing your transition into the world of RenderMan Studio.

Workspace

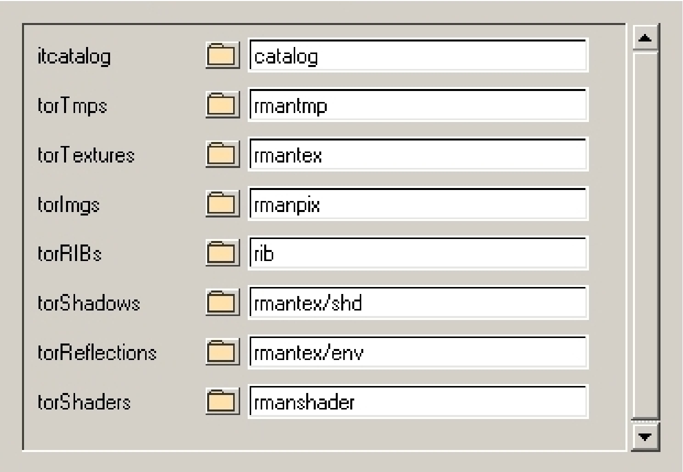
RenderMan Studio introduces a complete reconceptualization of the Workspace and the way that RenderMan resources are managed.

- Output directories now incorporate the new variable, `${STAGE}`. By default, `${STAGE}` resolves to your scene name; this is site- configurable via the `RMS.ini` file.
- Search Paths now resolve to `${PROJ}`, the workspace root. In addition to across-the-board improvements, this allows intra-project referencing to work with Slim and RenderMan for Maya Pro.

Additionally, `RMSExpression` has been exposed, granting users increased flexibility for site-specific configuration.

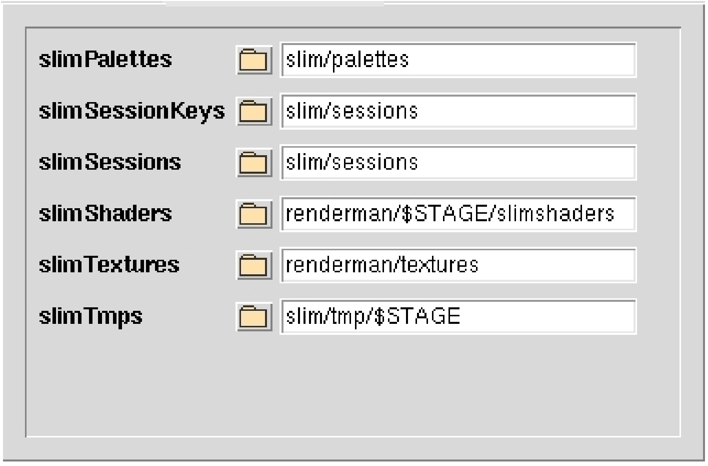
NOTE: In order to enjoy these new changes with existing projects, you will need to delete the existing `RATworkspace.ws` file from your project's root directory.

Here's a Before and After look at the default directory structure for Slim resources:



itcatalog	catalog
torTmps	rmantmp
torTextures	rmantex
torImgs	rmanpix
torRIBs	rib
torShadows	rmantex/shd
torReflections	rmantex/env
torShaders	rmanshader

The RAT Way



slimPalettes	slim/palettes
slimSessionKeys	slim/sessions
slimSessions	slim/sessions
slimShaders	renderman/$STAGE/slimshaders
slimTextures	renderman/textures
slimTmps	slim/tmp/$STAGE

The RMS Way

RenderMan for Maya also uses the *renderman* subdirectory of your project as the starting point for the rest of the renderer's resources. By default, the following can be found in **`$PROJ/renderman/$STAGE/`**:

data/$FRAME/

point clouds (.ptc), brick maps (.bkm), shadow maps (.tex), etc. are generated on a per-frame basis and deposited here.

images/

Final images (when rendered to file via Alfred or Mayabatch renders)

rib/$FRAME/

RIB files are generated on a per-frame basis and deposited here. In addition, per-job RIB files are deposited in `rib/job/`.

shaders/

Compiled shaders (.slo) from translated Maya materials are stored in this directory.

Separately, texture files (i.e. image files converted via `txmake`) are stored in **$PROJ/renderman/textures/**, along with Slim textures (see the screenshot above).

Slim

We strongly recommend consulting the Slim documentation for a complete rundown of Slim's functionality, much of which is new in version 7.0. Here's a quick rundown of a few important things:

Attach / Detach

Assigning appearances has been handed over to Maya. Appearances are created in Slim and then added to your Maya scene. The appearances are then attached to Maya nodes via the usual Maya ways, e.g. Hypershade, the Lighting/Shading menu, MEL, et cetera.

Ensemble Adaptors

Slim's Ensemble Adaptors are not supported by RenderMan for Maya. Instead, there is a special RenderMan Attribute that can be added to Maya Shading Groups via the Attributes menu of the Attribute Editor: ***Add Adaptation***. The adaptation attribute appears under the *Extra RenderMan Attributes* and gives users the same adaptive behavior as Ensemble Adaptors in RAT.

Light Shaders

Currently, there is no integrated workflow for adding Slim's light shaders to Maya scenes. There is, however, a workaround :

1. Build your light shader appearance in Slim.
2. Compile the shader (e.g. click on the preview swatch). This will create a `.slo` file in the appropriate `/slimshaders` directory.
3. Create a corresponding light in Maya.
4. Attach a *Custom Light Shader* via the Attributes menu for your light's Shape node: *Attributes-> RenderMan-> Add Custom Light Shader*.
5. In the *Extra RenderMan Attributes*, click on the widget to create a new RenderManLight node.
6. The *Shader* field in your new node allows you to browse to the aforementioned `.slo` file.

MapGen

Instead of *MapGen* nodes, users must create a Reference Pass.

RIBBox

RIBBox functionality is now achieved via Ri for MEL.

Archiver

Users can now use Maya's *Export* menu to export RIB archives.

TCLBox

There is no direct support for adding TCL via RfM. Users can employ TCL via MEL scripts, e.g. `rman tcl subst "tcl bits"` or `rman tcl eval "tcl whatnots"`.

Illumination

There are three critical differences in illumination techniques when you start using RenderMan Studio. First, as mentioned above, there is no integrated workflow for adding Slim Light shaders to your scene. Second, shadows are created by adding Shadow Attributes to Maya lights. And third, global and image-based illumination are now done via a RenderMan Environment Light.

For more information on the shadow workflow, please see the Deep Shadow tutorial. Further discussion of global illumination techniques using the Environment Light can be found in the Global Illumination tutorial.

Geometry

MTOR Subdivision Surfaces

MTOR subdivs just work . Existing subdivs are rendered properly by RfM and, because the MTOR subdiv plugin is unlicensed, in RMS 1.0 users can load the plugin and use it, happily and without incident, with the RfM. Creases, crease strength, corners, etc. are translated automagically .

RIB Archives

As mentioned above, RIB archives are created via Maya's Export menu in RenderMan for Maya. They are then referenced via a MEL script attached to, for example, a Maya locator.

In this case, select the Locator and attach a *Pre Shape MEL* via the Attributes menu. In the *Extra RenderMan Attributes*, enter `RiReadArchive "archive name";` in the window for the MEL script.

Rendering

The big difference is probably the most obvious: renders are now initiated via Maya's UI. A basic rundown is available in the Getting Started section of the documentation.

Here are a couple of additional notes on rendering with RenderMan Studio:

Spooling Renders via Alfred

Simple Alfred-spooled renders can be started via Maya's Batch Render UI. Selecting the *options* box gives users access to several different modes:

- `mayabatch local` (no RIB)
- `mayabatch remote` (no RIB)
- `immediate rib, local render`
- `deferred rib, remote render`
- `remote rib, remote render`

Note that the Mayabatch remote renders still require Alfserver to be installed on your render nodes.

The Batch Render UI also offers the usual complement of job options, such as *Frames per Server*, *Job Priority*, and *Renderer Arguments*.

Miscellaneous

Ray Tracing Features

Special ray tracing features (e.g. blur parameters for reflections and refractions) are accessed by adding controls to Maya material nodes.

Prev | Next

The Slim Crash Course

Creating a simple layered shader with *Delux*

This tutorial is a brief introduction to **Slim**, covering the basics of creating a shader. Many of the different aspects of using Slim will be covered during the creation of this simple shader, but for all the nitty gritty details, please consult the Slim Documentation.

To get started, start Maya and make sure that the RenderMan plugin is properly loaded.

1 CREATING A PALETTE

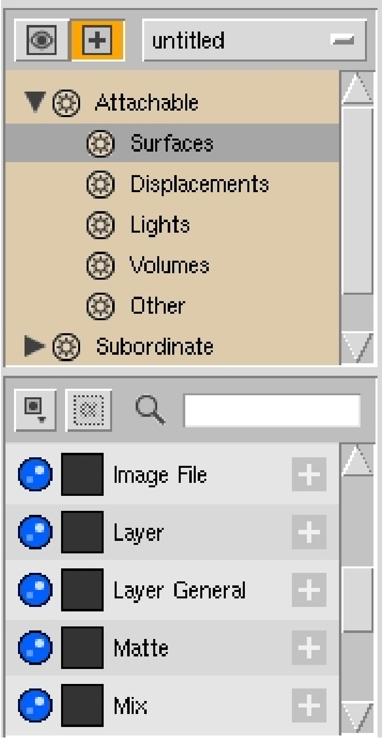
First things first start a Slim session. To open Slim from the Maya UI, go to the Maya *Window* menu and select **Rendering Editors > RenderMan > Slim)**

When you first open Slim you are in a default Session, with a default Palette. To create a new palette in the Slim window, select **New Palette** from the *File* menu. Alternatively, you can use a keyboard shortcut **Ctrl + N**. Once you're working with Slim regularly, you'll probably be working with existing palettes. You can open an existing Slim Palette from the *File* menu, too, or you can use the **Ctrl + O** keyboard shortcut. Sessions can contain multiple Palettes, and Palettes can be *internal* or *external*. For a more detailed look at Sessions and Palettes, see the Sessions and Palettes page of the Slim documentation.

2 CREATING AN APPEARANCE

Slim has two operational modes: *Browse* and *Create*. As you might guess, in order to create a new palette, you need to be in *Create* mode. Click on the plus-sign icon to switch to *Create* mode. The icon's background will change to a nice shade of orange, reminiscent of the flesh of a well-cooked butternut squash.

The primary building blocks for Slim are the Attachable appearances. These include surfaces, displacements, lights, and volumes. They are, simply, the appearances that you can attach to Maya nodes. In our case here, select *Attachable* > *Surfaces*, and then choose a **Layer** appearance.



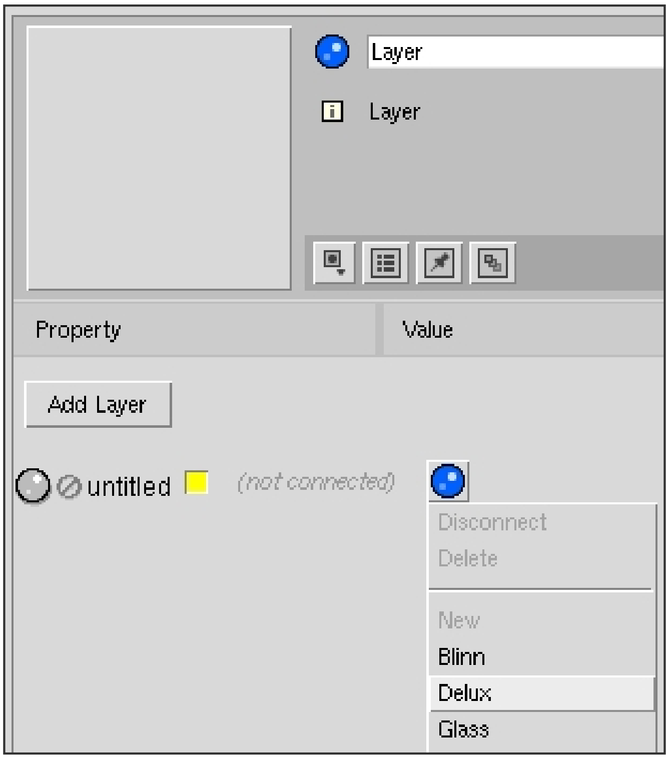
The Palette View

3 CREATING A LAYER

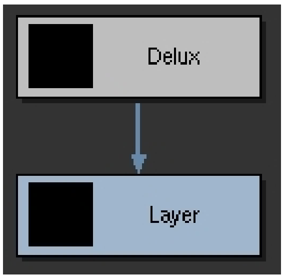
So we have, nominally, a Layer appearance. Of course, it has no layers, so it is basically an empty shell of an appearance. Let's add a layer)

Click on the *Add Layer* button.

Your appearance now has one layer, but it is an empty layer notice that it is telling you that it is *(not connected)*. Click on the blue widget and select a Delux shading model as your base layer.



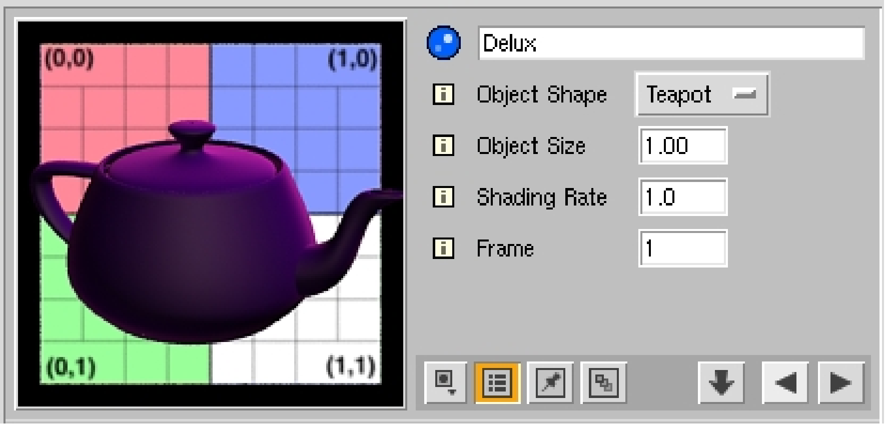
Adding a layer.



Note that you can now see the Delux node attached to your Layer appearance in the Graph View. Click on the Delux node (in either window) to open it in the Appearance View.

4 GETTING AROUND

Getting around in Slim is as simple as can be. You can move around between open palettes in the Palette View. Within an appearance you can choose different nodes right in the Graph View, or you can navigate linearly (think in terms of a Web browser) within the Appearance View using the *back* and *forward* buttons. You can also select any Downstream Appearance by clicking on the arrow to the left of the aforementioned buttons and selecting the appropriate node.



Delux

Object Shape: Teapot

Object Size: 1.00

Shading Rate: 1.0

Frame: 1

You might notice, in the image to your right, that there is quite a bit of information, and there are more than a few options for what you can see and how you see it. Be sure the check out the documentation on The Appearance View for all the sordid details. In this particular illustration, we're in the Delux, so you can click on the *back* button to get to the Layer (which you can also select from the arrow's list), or you can click on the *forward* button to get to the Spline that we've attached to the Diffuse component's Color parameter)

What's that? You say we haven't attached anything to any so-called Color parameter? Well, let's get to it then)

5 CREATING (AND TWEAKING) SHADING COMPONENTS

The Delux shading model is powerful because different surface charateristics (called shading components) can be added to the base Delux model. These additional components can be: specular, rim light, subsurface scattering, etc. Back in our Layer shader, open up the Delux layer and we'll throw in a couple of components.

1. First add a shading component by clicking on the **+** .
2. Next, select the component selector, ●, and choose *Rim*. This adds rim lighting control to the Delux. You can view the parameters by clicking on the triangle next to the **V2** label.
3. Create a second component, as shown above, and choose a Specular component.

Now our once diffuse model has specular highlights and rim lighting built directly into it!

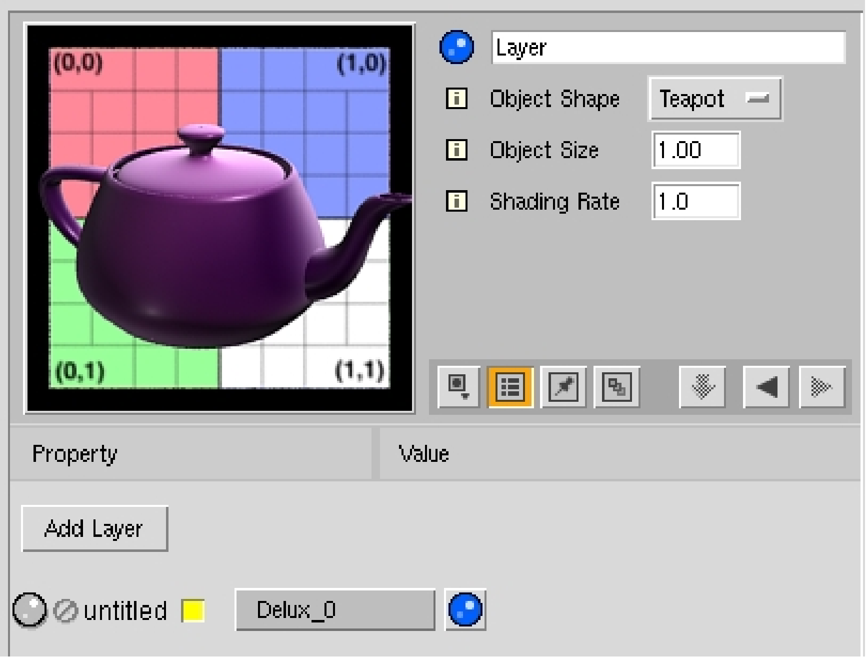
Now, that little Diffuse color tweak)

You have oodles of subordinate nodes to choose from when it comes to building Slim appearances. You can manipulate your shader's parameters by attaching a variety of texture maps, patterns (like noise, or fractals), or functions (including an SL Box, to insert raw shading language into your network). For our simple exercise, we're going to map a spline to the color parameter of our Delux's Diffuse component.

Click on the connection widget to the right of the Color parameter and select *Pattern-> Spline*. You'll notice that the connection appears in the Graph View, as well. Once again, you can click on the Spline in either view to bring it into focus in the Appearance View.

There are two important bits of the Spline pattern, as you'll notice. There's a *Pattern*, again, and a *Spline Color*. You can add additional control points to the spline by clicking within the gradient, and you can bring up the Color Picker by clicking on the round widget to the left of the gradient. Control points can be deleted by clicking on the square box to the gradient's right. Let's bring up the Color Picker and make our Spline a nice progression of pleasant shades of Purple, with a touch of Mauve or)something)

Now, if you click on the Pattern component of our Spline, you might notice something interesting: back when we clicked on the connection widget for the Color parameter, there were a slew of options, in terms of what we could connect to, but now our options are considerably fewer; Slim is smart enough to cull the inappropriate connections. Why don't we select *Pattern-> (Random Tiler - Worley)-> Wavelet Noise*?



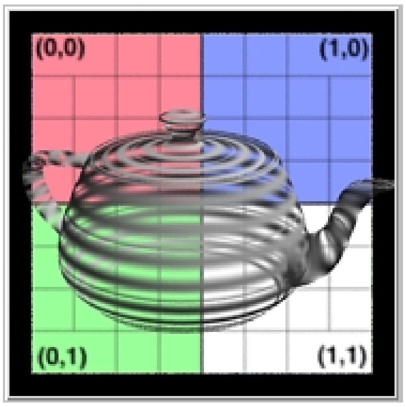
All that being done, let's go back up to the top level of our network and click on the Preview Swatch to compile our shader. Chances are good it will look at least a little bit like the picture just to the left of this text.

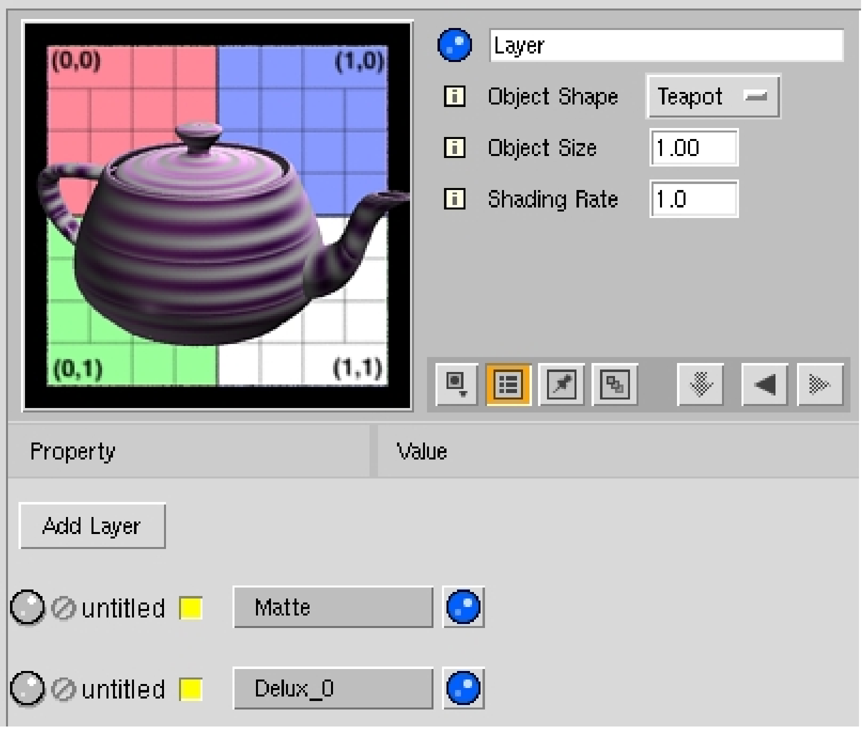
Now, that being all well and good, time to add another layer)

5 MORE LAYERS, PLEASE!

Okay, lets create some kind of funky, stripey mask to layer on top of our wavy swatch of purple and mauve. This is pretty simple, so we'll go through it fairly quickly)

1. In our Layer, click on the *Add Layer* button again.
2. For simplicity's sake, select a *Matte* appearance. This should not be confused with a Matte Object.
3. Click on the Matte appearance to bring it into a focus, then connect a *Ramp* to the *Opacity* parameter: *Pattern-> Ramp*.
4. Change the *RampType* to a *T Ramp* and enable tiling by clicking on the *Tile* checkbox.
5. Make the Ramp a series of alternating black and white stripes (completely transparent vs. completely opaque) and select an *ST* manifold. The manifold controls how a layer or function is mapped on the object in 3D space. (ST mapping is similar to UV mapping in Maya; note that there are a slew of manifold options, including a `MayaPlace2d` manifold.)
6. Click on the *ST* node and change the *T Repeats* to 2, either with the slider, or by entering the number 2 in the appropriate space.
7. Navigate your way back up to the Matte node and click on the swatch to compile the shader. It should bear an uncanny resemblance to the image at right.





Layer

Object Shape: Teapot

Object Size: 1.00

Shading Rate: 1.0

Property	Value

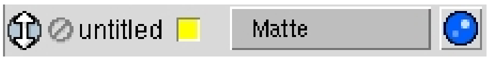
Add Layer

untitled 🟨 Matte

untitled 🟨 Delux_0

Now let's jump back up to the Layer node and click on the swatch to compile the whole shebang. What we get is, as you might have guessed by now, our Matte appearance layered on top of our Delux, which we can see clearly through the transparent bits of the Matte.

One more nice thing that you can do with layers: you can move them around. Click on the pretty, round widget to the left and you can drag a layer such that it is oriented on top of or under any other layer.



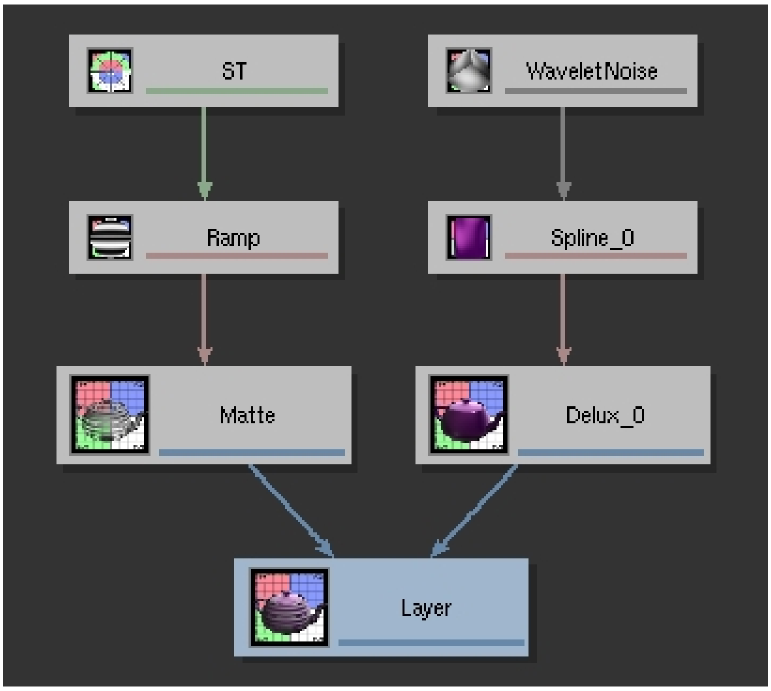
untitled 🟨 Matte

In this particular case, changing the order wouldn't make much sense, given that the Delux is completely opaque, but you can try it anyway, if you want)

6 CONCLUSION

Well, that's that. We've built ourselves a nice little Slim shader. See that picture over there →?
That's what our final Layer appearance should look like in the Graph View.

All that's left now is to select *Add to Scene* from the Appearance menu. **Note that you can retain control within the Maya UI over any given parameter in your nodes by clicking on its connection widget and setting it to use an *External Value*.** What's more, you can bring it all back into Slim *after* you've added your appearance to your scene simply by clicking on the *Edit in Slim* button in the Appearance Editor.



This tight integration between Slim and Maya offers you unprecedented flexibility as well as a simple way to expand your shading palette using appearances from Slim along with materials from Maya.

Please be sure to check out the Slim Documentation for more information, including template construction, Slim scripting, and more.

Prev | Next

Dynamic Read Archives

To get started, let's open the Maya scene, *teapot_anim.ma*.
(Where are the tutorial files?)

This tutorial demonstrates how to make a utility which allows an arbitrary RIB archive sequence to be referenced in a maya scene. The hope is that this is a useful utility which also illuminates several aspects of how RenderMan for Maya can be extended. First, we'll generate a sequence of RIB archives. Then through mel scripting we'll attach the sequence to a cube, which will serve as a standin for the geometry in the archive.

1 GENERATING RIB ARCHIVES

Open the scene called *teapot_anim.ma*. The file contains a teapot that walks in place.

- Load the RenderMan For Maya plugin.
- In the outliner, select the node called *teapot_solo:tea*.
- From the File menu select the options dialog of *Export Selection* and switch the file type to RIB_Archive .
  1. Scroll down to the file type specific options, enable *Multiple Frames*, and type in a starting and ending frame. Frames 1 through 60 cover about one cycle of the walk.
  2. Also enable the *Export Shaders* checkbox.
  3. Click the *Export Selection* button and browse to a folder where you want to stash the sequence, and type a name, like "teapot". The frame and rib extension will automatically be appended.

●Tip

`world_ref` has been implemented to embody a reference world coordinate system that's safe within archives for Maya projections, bumps, etc.

2 CREATE A STANDIN

Create a poly cube. Scale and translate it to enclose the original teapot. Then select *Modify->Freeze Transformations*; look in the Freeze Transformations options box to make sure translate, scale, and rotate are enabled. When we're done, the cube will serve as the teapot's standin bounding box, and when you render, the teapot will appear in the place of the box. During rendering, the teapot will be loaded into memory just in time whenever the bounding box of the cube is encountered.

At this point, you could template the teapot hierarchy and group it under the cube transform to allow for visualizing the teapot, or just delete it. We also need to prevent the cube from appearing in renders. This can be done by templating the cube shape node, or if you prefer to be able to manipulate the cube like regular geometry, the shading group should be disconnected from the cube shape.

Next, in order to tell RenderMan for Maya that it should load the teapot archive we need to add a few new attributes to the cube's transform node.

3 NEW SETTINGS

Let's put three new settings on our cube transform node that will allow us to control the archive name and sequence number.

- `draFile` A file browser control that lets us pick a rib archive file.
- `draUseSequenceNumber` A checkbox that indicates that the archive file should be treated as a sequence.
- `draSequenceNumber` A number that will be driven by an expression that indicates the current frame of the sequence.

RenderMan for Maya takes the approach of allowing extra attributes to be added to nodes, and they appear in the Attribute Editor under a section called *Extra RenderMan Attributes*. In order for our new attributes to appear under that section, we need to let RenderMan for Maya know about them by declaring them in its initialization file. RfM puts all of its declarations in files with a .rman extension. Let's put the following in a file called `rfm_dra.rman`⟩

```
        rman "-version 1" {
        Declare param {string draFile} {
                        label "Rib Archive"
                        description "The rib archive file, which may be the first in a
sequence."
                        subtype file
                        range {*.rib}
        }
        Declare param {int draUseSequenceNumber} {
                        label "Use Sequence Number"
                        description "Indicates that the Rib Archive file should be interpreted
as a sequence"
                        subtype masterswitch
                        range {draSequenceNumber}
        }
        Declare param {int draSequenceNumber} {
                        label "Sequence Number"
                        description "The sequence number can be an expression."
        }
        }
```

Put the file wherever you want to keep it, and then we'll tell RfM to load it during initialization. Open your RenderMan_for_Maya.ini file and add the following, replacing the path with the l ocation you chose.

```
        set myscripts "/path/to/my/scripts"
        LoadExtension rman [file join $myscripts rfm_dra.rman]
```

Next, some MEL scripting⟩

4 ADDING SETTINGS TO THE STANDIN

We need to add the three new settings to the standin cube, along with one other setting that RfM already knows about. That one is called `Post Transfom MEL` and it serves as an entry point for executing an arbitrary MEL script at the time that an object is rendered.

Here's a MEL script which adds our new settings to selected objects. You can put it in a file called `rmanAddDelayedReadArchiveAttrs.mel` and install it with your MEL scripts. Add a button to one of your shelves that invokes the script.

```
        global proc rmanAddDelayedReadArchiveAttrs()
        {
            string $selected[] = `ls -sl`;

            int $i, $j;
            for( $i=0; $i < size($selected); $i++ ) {
                // Generates a maya attr name, given a setting name as declared
                // in RfM's ini files.
                string $attr = `rmanGetAttrName "postTransformScript"`;
                rmanAddAttr $selected[$i] $attr "rmanOutputDelayedReadArchive";
                $attr = `rmanGetAttrName "draFile"`;
                rmanAddAttr $selected[$i] $attr "";
                $attr = `rmanGetAttrName "draUseSequenceNumber"`;
                rmanAddAttr $selected[$i] $attr "";
        $attr = `rmanGetAttrName "draSequenceNumber"`;
        rmanAddAttr $selected[$i] $attr "";
        // Create a default expression which sets the sequence number
        // to the current frame number.
        expression -s ($selected[$i] + "." + $attr + "=frame")
            -o $selected[$i] -ae 1 -uc all;
    }
}
```

5 LOOP THE SEQUENCE

Select the cube, and invoke the `rmanAddDelayedReadArchiveAttrs` script. Look at the shape node tab in the Attribute Editor. You should see four attributes under the *Extra RenderMan Attributes* section.

Start by pointing the `RIB Archive` attribute at one of the rib archives we exported earlier. Click on the folder icon next to the atrribute to bring up the file browser. Pick one of the rib files you exported.

Check the `Use Sequence Number` parameter. This will tell the script to substitute the frame number in the file name.

Right-Click on the "Sequence Number" setting, and select "edit expression". Our script above created an expression which sets the sequence number to correspond to the current frame. Since 60 frames represent one cycle of the walk, let's loop it. Change it to:

```
pCubeShape1.rman__param___draSequenceNumber=frame%60
```

Now

6 THE DELAYED READ ARCHIVE SCRIPT

Notice the `preShapeScript` setting is filled in with a MEL command called `rmanOutputDelayedReadArchive`. That's the script that is executed right before RfM goes to render the cube. We want it to magically make a RenderMan Interface call like:

```
RiProcedural "DelayedReadArchive" ribfilename-with-correct-sequence-number bounding-box-of-cube
```

RfM has handily defined RiProcedural as a MEL command, so our MEL script can compute the appropriate RIB file name, given the current sequence number, and query the bounding box of the cube. Here's the script. (You should put it in a file called `rmanOutputDelayedReadArchive.mel`, and install with your MEL scripts.

```
proc string replaceFrameNumber( string $str, int $f)
{
    string $newStr;
    string $buf[];
    tokenize( $str, ".", $buf );
    int $i, $j;
    for( $i=0; $i < size($buf); $i++ ) {
                // find substring that's a number
                if( `match "[0-9]+" $buf[$i]` == $buf[$i] ) {
                    string $fstr = $f;
                    string $pad = "";
                // pad with zeros
                for( $j=size($fstr); $j < size($buf[$i]); $j++ ) {
                                $pad += "0";
                }
                $fstr = $pad + $fstr;
                $newStr += $fstr;
```

```
                        } else {
                        $newStr += $buf[$i];
                        }
                        if( $i < size($buf)-1 ) {
                            $newStr += ".";
                        }
        }
        return $newStr;
        }

        global proc rmanOutputDelayedReadArchive()
        {
        // Find the object context within which this script is being called.
        string $object = `rman ctxGetObject`;

        // Get the value of the draFile attr.
        string $attr = `rmanGetAttrName "draFile"`;
        string $drafile = `getAttr ($object + "." + $attr)`;

        // Check whether the file should be treated as a sequence
        $attr = `rmanGetAttrName "draUseSequenceNumber"`;
        int $doSeq = `getAttr ($object + "." + $attr)`;

        if( $doSeq ) {
                        // Replace the frame number in the file name with the current frame.
                        $attr = `rmanGetAttrName "draSequenceNumber"`;
                        int $seqNum = `getAttr ($object + "." + $attr)`;
                        $drafile = `replaceFrameNumber $drafile $seqNum`;
        }

        // Get the bounding box from the maya shape node, which we'll assume
        // encompasses the RIB archive.  RenderMan wants the bbox to be in
        // centimeters in object space.
        string $curUnit = `currentUnit -q -linear`;
        currentUnit -linear "cm";  // temporarily change to centimeters
        float $bbSize[3] = `getAttr ($object + ".boundingBoxSize")`;
        float $bbMin[3], $bboxMax[3];
        $bbMin[0] = -$bbSize[0]/2;
        $bbMin[1] = -$bbSize[1]/2;
        $bbMin[2] = -$bbSize[2]/2;
        $bbMax[0] = $bbSize[0]/2;
        $bbMax[1] = $bbSize[1]/2;
        $bbMax[2] = $bbSize[2]/2;
        currentUnit -linear $curUnit;

        RiProcedural "DelayedReadArchive" $drafile $bbMin[0] $bbMax[0] $bbMin[1] $bbMax[1]
$bbMin[2] $bbMax[2];
        }
```

7 RENDER!

Finally, template or delete the original teapot, and render. Try instancing the cube, you can make an army of lockstep teapots.

8 WHAT ABOUT MOTION BLUR?

In the first step, the teapot archive sequence was created by selecting the top node of the hierarchy, and all the teapot pieces went into one archive per frame. If you repeat that step with motion blur enabled in the RenderMan Globals, motion blocks will automatically be written into the archives. Each animated transform or shape in the teapot hierarchy is sampled two times and placed inside a motion block. Note that transforms may be sampled up to six (6!) times if they have an extra `motionSamples` attribute. We want the motion blocks to live inside the archive because `DynamicReadArchives` can't be placed inside motion blocks (although this is allowed for regular `ReadArchive` calls). You still do need to enable motion blur in the globals in order for motion blocks in the archive to have any effect.

Prev | Next

Pixar Animation Studios

Custom Geometry

When RenderMan for Maya goes to render and encounters a plugin shape node in Maya, the shape is skipped unless you tell RfM what to do with it. This example demonstrates how you can make RfM recognize your custom shape and render it with a RenderMan procedural plugin. As our custom shape, we'll use the quadricShape from Maya's devkit, which is a plugin that draws your choice of a sphere, cylinder, disk, or partial disk. There are attributes on the node, like radius, height, and sweepangle which will be interpreted.

1 CREATE THE quadricShape.

On your platform of choice, locate the quadricShape plugin in the Maya devkit, and compile it. Check out the Maya documentation for information on compiling the plugin.

Once you have the quadricShape plugin, open the plug-in manager in Maya and load it up. Then create a shape, like this, in the Script Editor:

```
createNode quadricShape;
select -r quadric1;
sets -e -forceElement initialShadingGroup;
```

You should see a cylinder in the scene, and since it's initially created without a shading group, we've attached the default shading group. Try changing some of the attributes on the node. Notice that the quadric is missing from renders.

2 MEL SCRIPTING

Next, some MEL scripting is necessary to tell RfM about the procedural plugin, which we'll build later on.

- **Set up a searchpath for procedural plugins**

  It'll be handy to refer to our procedural plugin just by name, rather than with a full path. This MEL procedure outputs an RiOption which sets up a search path for plugins in the RenderMan Studio devkit. You can paste it into the Script Editor, for now, or install it among your MEL scripts.

```
global proc rmanOutputProceduralSearchpath()
{
    // This proc sets up a searchpath for procedural plugins.

    // On windows its necessary to load a different version of the procedural
    // plugin depending on whether it'll be running internally in RfM or
    // in the standalone prman renderer.  We can find out if this script is
    // being called during ribgen or internal rendering, and
    // refer to a different procedural path based on that.
    string $subdir = "";
    if( `about -nt` ) {
                if( `eval("rman ctxIsRib")` ) {
                    $subdir = "prman_plugins";
                } else {
                    $subdir = "rfm_plugins";
                }
    }

    // Change this path to correspond to the location where you put your compiled plugin.
    // Note on windows the path needs to be in UNC format.
    string $pluginpath = "//C/Program Files/Pixar/RenderMan-Studio-1.0-Maya8.5/
devkit/custom_geometry/";
    RiOption "searchpath" "string procedural" ($pluginpath+$subdir);
}
```

NOTE: The path referenced in the examples above is Windows-specific. OS X and Linux should change it accordingly.

We can cause the searchpath option to be output into the RIB stream at the appropriate time, by invoking this procedure in the **Default RiOptions** MEL script, which is located in the *Render Settings* under the *Common* tab. It's not exactly common, but it's in the same vein as the Pre/Post Render scripts, so they're grouped together. Paste a call to the procedural above in the Default RiOptions MEL field:

```
rmanOutputProceduralSearchpath
```

- **Set up the `RiProcedural` Call**

  When RfM encounters the quadricShape while rendering, we'll run a script that inspects some of the attributes on the node and issues a `RiProcedural` call with an appropriate bounding box and argument list. The following proc does that. You can paste it into the Script Editor, for now, or install it with your MEL scripts.

```
global proc rmanOutputQuadricProcedural()
{
    // Find out which object this script is associated with
    string $object = `rman ctxGetObject`;
    if( `nodeType $object` != "quadricShape" ) {
                warning("rmanOutputQuadricProcedural can only operate on nodes of
type quadricShape\n");
                return;
        }

        // These are the attributes from the quadricShape node that our
        // quadric procedural plugin will interpret.
        int $shapetype = `getAttr ($object+".shapeType")`;
        float $radius1 = `getAttr ($object+".radius1")`;
        float $radius2 = `getAttr ($object+".radius2")`;
        float $height = `getAttr ($object+".height")`;
        float $startAngle = `getAttr ($object+".startAngle")`;
        float $sweepAngle = `getAttr ($object+".sweepAngle")`;

        // Assemble the values of the attributes into an arg list,
        // to pass to the procedural
        string $args = ($shapetype + " " + $radius1 + " " + $radius2 + " " + $height + " "
+ $startAngle + " " + $sweepAngle);


        // The RiProcedural needs to be supplied a bounding box, and since
        // the quadricShape node takes care of updating its bounding box,
        // we'll use that.  Note the RiProcedural needs its bounding box
        // to be expressed in centimeters.
        string $curUnit = `currentUnit -q -linear`;
        currentUnit -linear "cm";  // temporarily change to centimeters
        float $bbSize[3] = `getAttr ($object + ".boundingBoxSize")`;
        float $bbMin[3], $bboxMax[3];
        $bbMin[0] = -$bbSize[0]/2;
        $bbMin[1] = -$bbSize[1]/2;
        $bbMin[2] = -$bbSize[2]/2;
        $bbMax[0] = $bbSize[0]/2;
        $bbMax[1] = $bbSize[1]/2;
        $bbMax[2] = $bbSize[2]/2;
        currentUnit -linear $curUnit;

        // Issue the RiProcedural call, passing the bounding box and arg list
        RiProcedural "DynamicLoad" "quadricProcedural.dll" $bbMin[0] $bbMax[0] $bbMin[1]
$bbMax[1] $bbMin[2] $bbMax[2] $args;
}
```

NOTE: The .dll file referenced in the example above is Windows-specific. OS X and Linux would use `.so`.

RfM needs to be told to invoke this procedure when it encounters the quadricShape node. That's accomplished with a preShape MEL script. Select the quadricShape node, and in the Attribute Editor make sure the quadricShape tab is selected. Open the RenderMan Attributes manager via **Attributes->RenderMan->Manage Attributes**. From the list choose the *Pre Shape MEL* attribute and add it. The attribute shows up at the bottom of the Attribute Editor. Paste a call to the above procedural as its value:

```
        rmanOutputQuadricProcedural
```

3 THE QUADRIC PROCEDURAL

Now the maya scene is set up. Next, we'll build the procedural quadric plugin, which will be dynamically loaded when its bounding box is encountered during rendering.

The code and makefile for the quadric procedural is located in the RenderMan Studio devkit, which is included as part of your RenderMan Studio installation, in the *devkit* directory.

- **Set up your build environment**

  Proper compilation of the quadric procedural assumes you have two variables set in your environment: RMANTREE and RATTREE. RATTREE should be set to the path of your RMS installation, and RMANTREE can be set to either the location of your RenderMan Pro Server installation or `Path/to/RMS/rmantree`.

- **Build the plugin**

  Open a shell and change to the directory of the RMS devkit. There's an example makefile for Windows, which you can execute by typing:

  ```
          make -f Makefile.windows
  ```

  Or you can compile using the standard RenderMan compilation method. Use the standard C++ compiler to generate an object (.o/.obj) file, then generate a shared object (.so/.dll) file from the object file. Remember that, though using C++, you *must* use C style linkage. You also must ensure that your C++ compiler and libraries are compatible with the compiler and runtime libraries used by *PRMan* (gcc for Linux and OS-X and Microsoft Visual C for Windows). Here are example commands for building the plugin on several architectures:

  ```
      Linux:      g++ -fPIC -I$RMANTREE/include -c quadricProcedural.cpp
                  g++ -shared quadricProcedural.o -o quadricProcedural.so
      Mac OS-X:   g++ -I$RMANTREE/include -c quadricProcedural.cpp
                  setenv MACOSX_DEPLOYMENT_TARGET 10.3
                  g++ -bundle -undefined dynamic_lookup quadricProcedural.o -o quadricProcedural.so
      Windows:    cl -nologo -MT -I%RMANTREE%\include -c quadricProcedural.cpp
                  link -nologo -DLL -out:quadricProcedural.dll quadricProcedural.obj %RMANTREE%\lib
      \prman.lib
  ```

- **`quadricProcedural.cpp`**

  The code for the quadric is in `quadricProcedural.cpp`, located in the devkit directory. Procedural plugins must implement three entrypoints: `ConvertParameters`, `Subdivide`, and `Free`.

  You'll see prototypes for these exported at the beginning of the file:

  ```
  extern "C" {
          export RtPointer ConvertParameters(RtString paramstr);
          export RtVoid Subdivide(RtPointer data, RtFloat detail);
          export RtVoid Free(RtPointer data);
  }
  ```

  The `ConvertParameters` method takes the arg string that we passed in the RiProcedural call and stashes the values in a data structure, which it returns a blind pointer to. Our data goes into a struct with a value for each attr:

  ```
  enum QuadricsType {
          kCylinder,
          kDisk,
          kPartialDisk,
          kSphere
  };

  struct quadricsData {
      QuadricsType shapeType;
      RtFloat radius1;
  ```

```
    RtFloat radius2;
    RtFloat height;
    RtFloat startAngle;
    RtFloat sweepAngle;
};

export RtPointer ConvertParameters(RtString paramstr) {
    struct quadricsData* data;
    char* parameters;
    char* token, *tokenstate;
    int i, j;

    parameters = strdup(paramstr);

        data = reinterpret_cast(malloc(sizeof(struct quadricsData)));

        token = strtok_r(parameters, " ", &tokenstate);
    data->shapeType = static_cast(atoi(token));

    token = strtok_r(0, " ", &tokenstate);
    data->radius1 = atof(token);

    token = strtok_r(0, " ", &tokenstate);
    data->radius2 = atof(token);

    token = strtok_r(0, " ", &tokenstate);
    data->height = atof(token);

    token = strtok_r(0, " ", &tokenstate);
    data->startAngle = atof(token);

    token = strtok_r(0, " ", &tokenstate);
    data->sweepAngle = atof(token);

    free(parameters);
    return (RtPointer)(data);
}
```

The Subdivide method is where the quadric shape gets rendered. The cylinder, disk, and partial disk can each be represented by an *RiHyperboloid* primitive, and the sphere by an *RiSphere* primitive. These look somewhat different than the quadricShape in the Maya render view because it uses OpenGL primitive calls. RenderMan's primitives aren't polygonal, so they'll appear smooth.

```
export RtVoid Subdivide(RtPointer datap, RtFloat detail) {
        struct quadricsData* data = reinterpret_cast(datap);
        int i, j;

        RiTransformBegin();

        RtPoint p1;
        RtPoint p2;
        p1[0] = data->radius1;
        p2[0] = data->radius2;
        p1[1] = 0;
        p1[2] = 0;
        p2[1] = 0;
        p2[2] = 0;

        switch (data->shapeType) {
        case kCylinder:
                p2[2] = data->height;
                RiHyperboloid(p1, p2, 360, RI_NULL);
                break;
        case kDisk:
                RiHyperboloid(p1, p2, 360, RI_NULL);
                break;
        case kPartialDisk:
                RiRotate(90, 0, 0, 1);
                RiRotate(-data->startAngle, 0, 0, 1);
                RiHyperboloid(p1, p2, -data->sweepAngle, RI_NULL);
                break;
        case kSphere:
                RiSphere(data->radius1, -data->radius1, data->radius1, 360, RI_NULL);
                break;
        }
        RiTransformEnd();
}
```

And finally, the `Free` method frees up the struct allocated in the `ConvertParameters` method.

```
export RtVoid Free(RtPointer datap) {
        struct quadricsData* data = reinterpret_cast(datap);
        free(data);
}
```

4 RENDER!

At this point, you should be able to render with RenderMan for Maya. If you don't see the quadric in the render, the most likely problem is that the `searchpath` isn't set correctly. You should see the quadric shape in both internal RfM renders and spooled RIB renders, assuming that the location of the quadricProcedural is accessible from the machine where the spooled render is happening.

5 BUT CAN WE MOTION BLUR?

So you want to motion blur your custom shape, eh? When the transform for the custom shape node is animated, motion blur happens automatically. Your procedural doesn't need to do anything special. However, if your shape node deforms and you want to motion blur that, you'll need to do some extra work.

- **Cache Shape MEL Script**

  Normally, when RfM encounters deforming shapes, and motion blur is enabled, it will cache the state of the shape at Shutter Open and Shutter Close times. Then at render time both samples are output, surrounded by a motion block. RfM allows you to cache your own shape during those caching passes. You can specify a caching MEL script by adding an attribute called *Cache Shape MEL* to your shape. You may also need to add the *Evaluation Frequency* attribute, and change it to *Frame*. That lets RfM know your shape is deforming, even if its usual method of looking for upstream animated nodes isn't adequate for detecting deformation of the custom shape.

Prev | Next

Pixar Animation Studios

Introduction to it Scripting

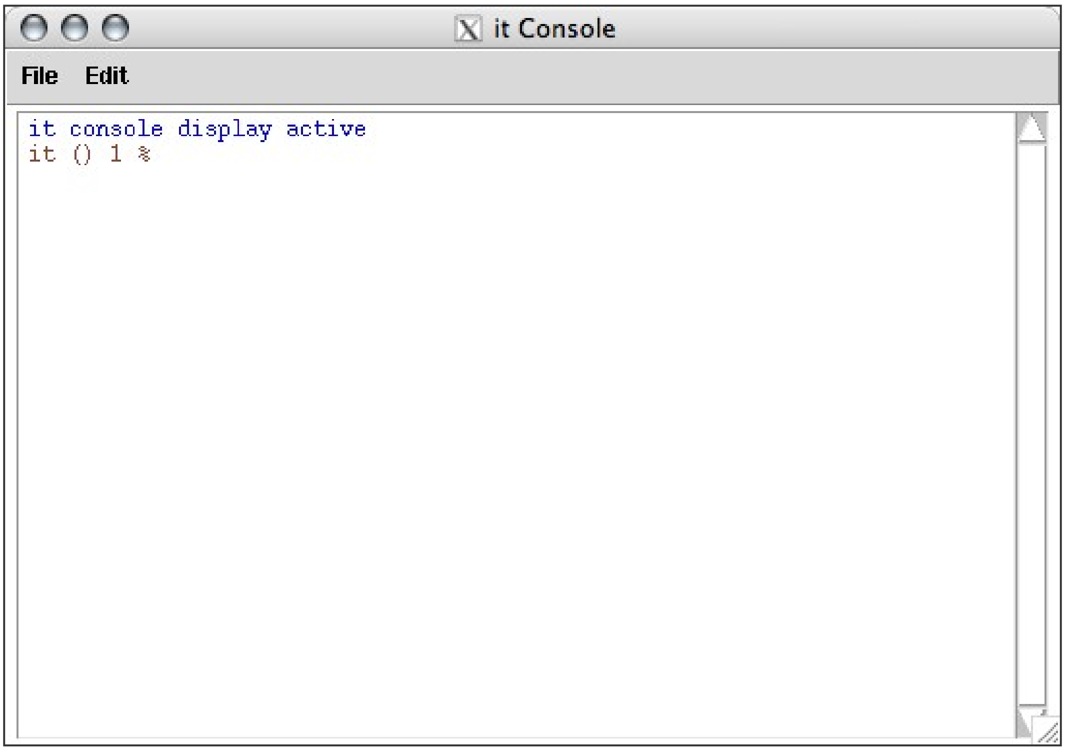
For this tutorial, you don't need a Maya scene file. All you need is a directory with a couple of images to play with.

1 STARTING it

Scripting it is done via the it Console. You can access the Console with it started via the Hub by right-clicking and selecting *Windows > Console)* or via the display window by right-clicking and selecting *Console)*

You can also start it via the command line, assuming it is in your path, and can open the Console directly using the -console flag:

```
% it -console
```



```
it console display active
it () 1 %
```

The it Console

2 LOADING AN IMAGE

1. In the Console window, use the cd command to navigate to a directory full of images.
2. Open the image by typing the following command in the Console window:

   ```
   ::ice::load image.tif
   ```

3. If you look at the it catalog, the image will appear with a _load handle. You can specify a handle using the -h flag *after* the file name, e.g.

   ```
   ::ice::load image.tif -h _it1
   ```

4. You can open more than one image at a time using the command this way:

   ```
   ::ice::load image.*.tif
   ```

Tip

The console has a convinient feature to make typing these commands easier called "tab completion". It works on both command and file/directory names.

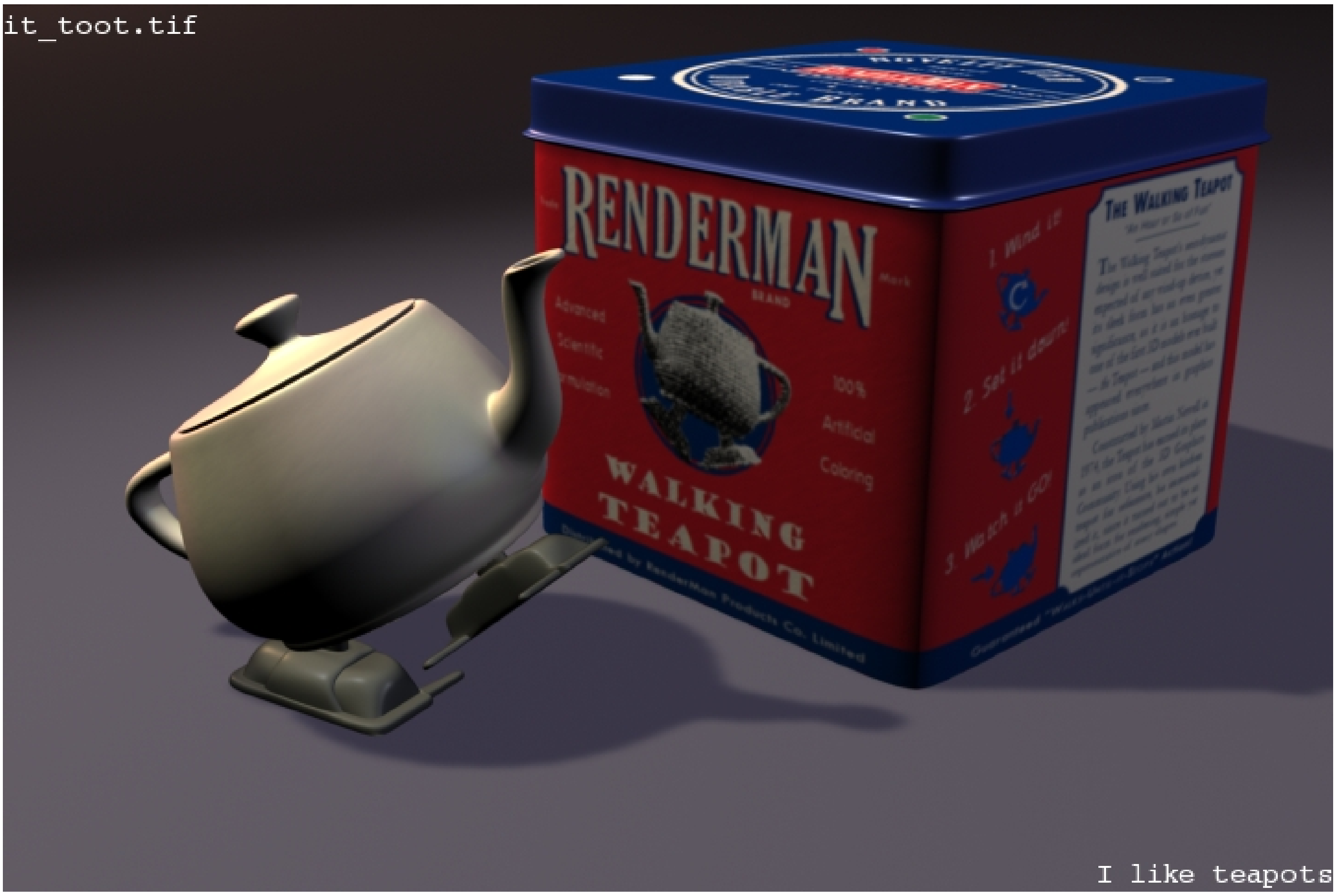
- Type `ice::` and the press the **tab** key. You get a list of all the functions that start with `::ice`. You'll see a few commands in there that we've included with it . It's easy to add your own.
- Type `ice::loadhe` and press **tab** and then return. You get the idea.
- Say you had a file called *image.345.tif* type `ice::load image` and press **tab**. The console will complete the file name.

Something else to make life easier in the Console are the arrow keys. When the blinking | cursor is on the last line (where you'd normally expect to type) the up and down keys let you cycle through all the commands you've typed. Previous commands can be edited this way.

Try another one of the included commands. This one shows you how you could use IceMan's text rendering to label your images.

1. Load an image that's at least 720x480
2. Type

```
::ice::slate -topleft %f -bottomright "I like teapots"
```



An it slate

`ice::slate` has other arguments, which you can find out about by typing `ice::slatehelp`.

If you are interested in modifying or making up your own functions you'll find the source to all the ice:: script in the /lib/it directory in the file named `TclIceExtensions.tcl`. If you write your own, you can put them in another file with a .tcl extension in that directory.

3 PLAYING WITH IMAGES

Now that you're comfortable getting to and running scripts in the Console you can start to explore all the functions that IceMan has. Let's say you have two images and you want to find the differences between them. Arithmetic on images in IceMan is easy!

1. Load two images, labeling the first one _it1 and the second _it2.
2. To compute the difference, from the console type:

```
it IceExpr "diff := _it2 - _it1"
```

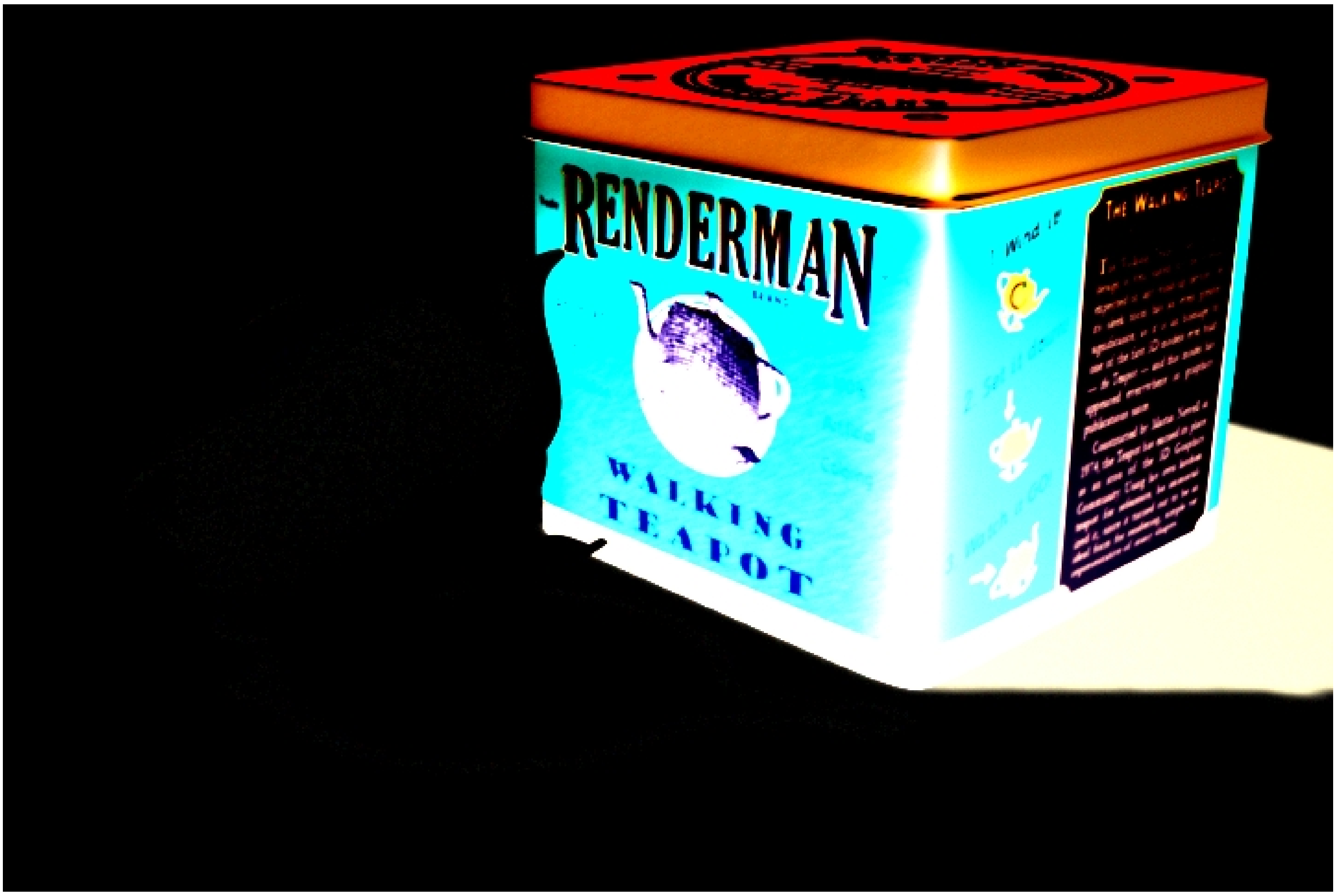
3. The resulting image is displayed and shows up in your catalog as `<script result>`, labeled `diff`.

A diff result

4. Let's say that the differences were really small, and you can't quite see them, so you want to multiply them by some factor. This is where using the "up" arrow becomes useful. Press that precious little button and then edit the previous command to read:

```
it IceExpr "diff := (_it2 - _it1) * IceColor(10)"
```

When you hit return it will update the "diff" image (you just lost the first one, unless you saved it).



A diff operation, multiplied

You might be wondering what `IceColor(10)` did above. The short answer is it made a special kind of image called a Card which is conceptually infinite in size and has the same value at every pixel. Whenever you need an image that has the same value everywhere use a Card.

Having gone through that operation the long way, it's worth mentioning that there is a shortcut. Looking at differences is so helpful, we included `::ice::diff` to save you the all that typing. In the example above, our initial operation could be performed with this simple command:

```
  ::ice::diff _it2 _it1
```

5 SUMMARY

it and IceMan provide users with a seriously robust scripting language for manipulating images, without getting overly complex. Take a look at the it documentation for a look at the bigger picture, so to speak. Be sure to check out the IceMan Reference Guide for a complete rundown of what you can do via scripting. And don't be shy about trying out the other it scripting tutorials, which take you throughbuilding a Web page via script and combining tcl and IceMan scripts.

More Fun with "it" Scripting

Once you've run through the basics, you're ready to delve a little deeper into scripting "it". In this tutorial we'll go over the steps to make a script that will create a Web page out of a catalog, including making thumbnails and images notes. This will demostrate the two major objects available in the scripting environment; the Catalog and the Image.

As with the introductory tutorial, you don't need a Maya scene file for this. All you need is a directory full of images to play with.

1 ADDING AN EXTENSION

First, let's set up a new extension script file. This is a file that will get sourced into "it"'s brain each time "it" starts.

All of the tools in RenderMan Studio use initialization files (.ini) that can be customized by the user. *We strongly recommend that you do not edit the .ini files in the installation*; instead, create supplemental .ini files and place them in a directory that you can point at with the $RMS_SCRIPT_PATHS environment variable. In this case, if you don't already have one, create an `it.ini` file and add the following line:

```
LoadExtension tcl /path/to/your/scripts/Web.tcl
```

Needless to say, adjust the path accordingly for your pipeline.

Next, let's verify that the extension is getting loaded properly. Create the aforementioned `Web.tcl` file and put the following in it:

```
RAT::LogMsg COPIOUS "Web Page Extension"
```

Make sure you have saved both files (the .ini and .tcl) and then launch "it". Once "it" is open, go to the *Message Log* window (right click in the *hub*, then select **Windows>Message Log**). Change the *Message Filter* to its second-most verbose setting, which is COPIOUS . You will see all the files that were loaded as "it" started, ending with our new one: `Web.tcl`. You can also see that Web.tcl produced something of its own: the message Web Page Extension .

2 GETTING YOUR SCRIPT ON

Now lets make our script actually make a small Web page. The important tasks this script is performing are:

1. Getting the current catalog
2. Getting a list of all the images in that catalog
3. For every image we find out its name

Here's what our simple Web page script looks like:

```
proc makeWebPage {fileName} {

   set f [open $fileName w]
```

```
       puts $f "<html>"

       set cat [it GetCurrentCatalog]

       set imgList [$cat GetImageList]
       puts $f "<ul>"
       foreach img $imgList {
           set name [file tail [$img GetFileName]]
           puts $f "<li>$name</li>"
       }
       puts $f "</ul>"
       puts $f "</html>"
       close $f
    }
```

To run this, first load a few images in "it". Open the "it" console and run the script with the following command:

```
makeWebPage /tmp/index.html
```

The script will generate a simple bulleted list of the images loaded in our catalog.

●Tip

You can use the *Save Session)* and *Restore Session)* functions here to help you get a catalog set up quickly.

3 WEB BLING!

What say we trick out our little Web page? Let's make a dazzling four column table with thumbnail images of the contents. This will demostrate how to perform some basic image processing, including resizing the images in our catalog and saving the thumbnails to disk.

To create the thumbnail images we use the handy *Reformat* operator. *Reformat* can change and/or resize an image in many different ways, depending on how you want to crop or squeeze your images into the new shape. For the Web page we want to createm uniform-sized images that are letterboxed if they are not the right shape. In *Reformat* terminology that means preserve aspect ratio and don't crop .

●Tip

As it happens, "it" images can be annotated. Right-click in an image window and select *Notes* to reveal the note display/editor. When you save a session the notes get saved along with it. Before you run the example, add notes to a few of the images in your catalog to see how that will come out.

When you run the script this time you'll see that a directory called *thumbs* is created alongside the html file, and in the small jpeg versions of the images are saved therin. You'll also see in the "it" hub window that the Catalog now has a new image for each thumbnail that was made. You can delete them if you like.

So, without further ado, here's the tricked-out script:

```
RAT::LogMsg COPIOUS "Defining Web Page Extension"

proc makeWebPage {fileName} {

   set f [open $fileName w]

   set topDir [file dirname $fileName]
   set thumbDir [file join $topDir thumbs]
   if ![file isdirectory $thumbDir] {
       file mkdir $thumbDir
   }

   puts $f "<html>"
   puts $f {<table cellspacing="10" align="center">}

   set cat [it GetCurrentCatalog]
   set imgList [$cat GetImageList]

   set col 0
   foreach img $imgList {

       # do we need an new row?
       if {$col == 0} {
           puts $f "<tr>"
       }

       # the start of a table cell
       puts $f "<td valign=\"top\">"

   # retrieve some basic information about the image
   set name [file tail [$img GetFilename]]
   set h [file tail [$img GetHandle]]

       # this is where we'll save the thumbnail image to
       set thumbFile [file join $thumbDir $h.jpg]

       # create an expression to make the image a fixed
       # size an to save it to disk as jpeg. The image will be at
       # maximum 200x200. Reformat takes care of any letter-boxing
       # that might be required
       set thumbExpr "_thumb.$h := $h Reformat(list(0,0,200,200), 1, 0);"
       append thumbExpr "_thumb.$h Save(\"$thumbFile\", IceOutputType Jpeg)"

       # "run" the expr
       it IceExpr $thumbExpr

       # put the image in the page using a page relative
       # path name
```

```
                puts $f "<img src=\"thumbs/$h.jpg\" alt=\"$name\">"
                puts $f "<br><b>$name</b>"

                # if the image has notes put them in too
                set notes [$img GetNotes]
                if ![string equal $notes {}] {
                   puts $f "<small><pre>\n$notes</pre></small>"
                }

                # the end of a table cell
                puts $f "</td>"

                incr col
                 # did we get to the end of a row?
                if {$col >= 4} {
                    puts $f "</tr>"
                    set col 0
                }
            }

            if {$col != 0} {
                # there was an incomplete last row
                puts $f "</tr>"
            }

            puts $f "</table>"
            puts $f "</html>"
            close $f
        }
```

Replace the original script with the script above, save the file, and execute as you did in Step 2. This time you'll get a slightly spiffier page with thumbnails, notes, and labels.

4 MORE BLING!

There's one more cool trick we can add to this example. Make sure "it" is not already running, and that your PATH includes $RMSTREE/bin (assuming your RMSTREE is properly set), then open a command prompt window and find a directory containing 10 or so images. Now let's make a file called `mkpage.tcl` contaning the single line:

```
makeWebPage /some/path/index.html
```

(Change the path to whatever works best for you.)

Now at the command line type:

```
it *.tif -script mkpage.tcl
```

Sit back and watch as this will cause "it" launch, open all the files ending in .tif as images, and then run the script in mkpage.tcl. In the event you don't care to see the progression of the images open before your very eyes, you can throw in the -hide option on the command line above. Note that "it" reads the images and `-script` arguments in left to right order, which is why the `-script` had to come after the image names.

Combining Tcl with IceMan Scripting

In the previous examples we wrote the entire script in tcl and, as necessary, built one-line IceMan scripts to perform imaging operations. In this tutorial we'll get a little more complex and show how to write IceMan extensions, highlighting where you would choose one environment over the other.

The goal is create a command called *bracket* that would take a high dynamic range image (HDRI) and create a series of images that would represent mapping that HDRI at a variety of exposure values. Instead of outputting a new image for each exposure level, we are going to create a montage image containing all the results.

We're going to use the crop window, which the user drags out in the GUI. The crop window is only available at the tcl level. We're also going to be iterating over an image and shuffling about the results and creating a montage, something IceMan scripts excel at. We'll try to keep the bits that each environment does well in the right place.

As with the introductory tutorial, you don't need a Maya scene file for this. All you need is an image to run the script on. An HDRI image will show off the particular utility of this script, but modifying the example to perform any other image processing operation should be easy.

1 ADDING AN EXTENSION

First, let's set up the new extension script files. In this example we'll be adding both a `tcl` extension and an `io` one. See the previousexample on how to set up a local user *it.ini* file. This time, add two new lines to *it.ini*, one for each script file we'll be working on.

```
LoadExtension tcl /path/to/your/scripts/bracket.tcl
LoadExtension io /path/to/your/scripts/bracket.io
```

Change the path according to where you actually put your scripts.

2 THE TCL PART

Now we'll concentrate on the tcl part. This will retrieve the current crop window from the GUI and process any arguments. We use a standard extension *ice::getopt* to pull out any optional arguments. Tcl routines can have a variable number of arguments if their last formal argument is called *args*. We adopt a convention of always having arguments specified in the form `-flag <value>`. This simplifies processing arguments that might be lists, or strings that might have embedded spaces.

So here's *bracket.tcl*:

```
# bracket - create a montage of various exposure levels of the current
#           image operating on the current crop region. Exposures are
#           made with the ExrToneMap operator.
#
# elow   - initial exposure value
# ehigh  - maximum exposure value
# nsteps - the number of brackets to make
# args   - optional args:
    #  -o <output> output image name (default _montage)
    #  -i <input>  input image name (default current image)
    proc bracket {elow ehigh nsteps args} {

        set cat [it GetCurrentCatalog]

        # process any optional args
        array set argmap $args
        set i [ice::getopt argmap -i {}]

        if [string equal $i {}] {
        set img [$cat GetCurrentImage]
        # don't bother continuing if we can't find an image
        if [string equal {} $img] {
            error "no current image"
        }
            set i [$img GetHandle]
        }

        # unless specified the output image's handle will be "_montage"
        # users can specify -o <handle name> to override this.
        set o [ice::getopt argmap -o _montage]

        # crop is a string, we let the io code parse it
        set crop [$cat GetCropRegion -fmt Ice]
        it IceExpr "$o := bracket($i,\"$crop\",$elow,$ehigh,$nsteps)"
    }
```

Notice how *ice::getopt* works: if the user doesn't use a -o option the output of this script will be called _montage . A catalog always has a crop region; if none has been dragged out in the GUI it defaults to the entire image. The last line of the script calls and IceMan script with a routine called *bracket*. This doesn't exist yet, so we'd better create that now. First we'll just create a stub to get this going and test the tcl part. So enter the following in your *bracket.io* file:

```
  bracket := method(in, cropStr, elow, ehigh, nsteps,

      // tcl is very string oriented, io has numbers as
      // first class entities. Its also good at lists
      crop := cropStr split map(s, s asNumber)
  )
```

This gives you a first look at how to create a IceMan script routine. As mentioned elsewhere, the IceMan script language is based on Io. All this method is doing is taking the *cropStr* argument and splitting it up, which returns a list, and then uses *map* to turn all the elements of the list into Numbers. More information on the Io language can be found here.

Now to check all that works start it , load an image and then, in the console window, run it with:

```
  bracket 0 3 9
```

If you open the *Message Log* window and set the filter level to *Debug* you'll be able to see what your script is doing. If you drag out a crop window on the image first and then run your script you'll see the crop window coordinates being passed down to the IceMan script.

3 THE ICEMAN PART

Time to make the IceMan part do something for real. Change *bracket.io* to:

```
    bracket := method(in, cropStr, elow, ehigh, nsteps,

        // tcl is very string oriented, io has numbers as
        // first class entities. Its also good at lists
        crop := cropStr split map(s, s asNumber)

        // calculate the size of the montage image. We're going
        // to separate each element of the bracket by a 20 pixel
        // border and we're always going to do five small images
        // across
        cols := 5
        border := 20
        cWidth := crop at(2)
        cHeight := crop at(3)
        mWidth := ((cWidth + border) * cols) + border
        rows := (nsteps/cols + 0.5) roundDown
        mHeight := (cHeight + border) * rows + border

        outBox := list(0,0, mWidth, mHeight)
        // make a big image to put all the little copies into
        result := IceImage FilledImage(in ComponentType, in Ply, outBox, list(0))

        // crop out the part of the source image we're going to work on
        src := in SubImage(crop)

        e := elow
        for (r, 0, rows-1,
            for (c, 0, cols-1,
                // translate the cropped image and then "tone map" it.
                offset := list(border+c*(border+cWidth),border+r*(border+cHeight))
                small := src Move(offset)
                small := small ExrToneMap(e, 0, 0, 5, 2.2)

                result := result CopyFrom(small, small DataBox)

                if (e == ehigh, break)

                e := e + ((ehigh - elow) / nsteps)
            )
        )

        return result
    )
```

A couple of important things to note here: first, notice how in the loop we keep referring back to the cropped portion of the original image in the variable *src*. No need to make explicit copies of this image; IceMan handles all that for you. Also note how we can use the result of an operation (like *ExrToneMap* above) and assign it back to the same variable. Since you are no doubt wondering, IceMan keeps track of all these images and when they go out of scope they get released. No need to worry about deleting these intermediate images; life is good!

Now to try this out, run it , load up an image, and drag out a smallish crop window, say about 100 pixels square. Then, in the console window, type:

```
        bracket -1 3 9
```

You should see a nice montage of your crop window at various exposure levels. Cool⟩you can show this to the lighting director to get her to buy off on one of the levels.

Now if you've ever had to do this in a darkened screening room with who knows who else in the room, causing distractions, you'll know that sinking feeling when the director says, The one second to the left⟩ and then, as she walks out the door, says, Actually no, the other left.

Right then you'd wished the battery in your laser pointer hadn't just died. Well, we have another solution up our sleeves! We'll use IceMan's oh-so-handy text rendering feature to put labels onto each exposure bracket, spelling out which value was used. No mistakes in our reviews⟩

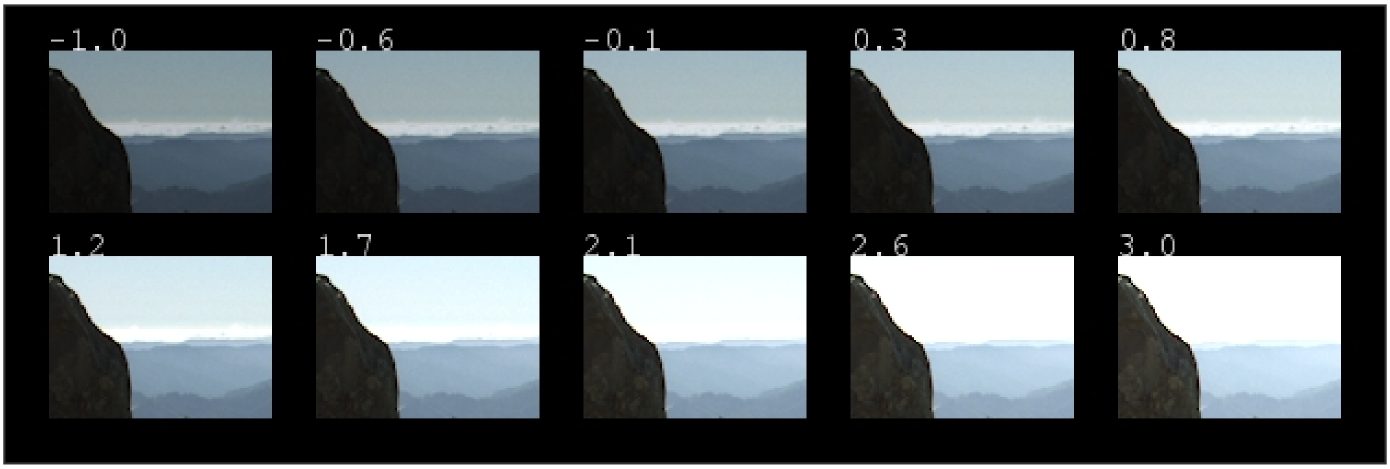
Here's the new version of *bracket.io* with text labels

```
    bracket := method(in, cropStr, elow, ehigh, nsteps,

        // tcl is very string oriented, io has numbers as
        // first class entities. Its also good at lists
        crop := cropStr split map(s, s asNumber)

        // calculate the size of the montage image. We're going
        // to seperate each element of the bracket by a 20 pixel
        // border and we're always going to do five small images
        // across.
        // Hard code some parameters we might later pass as arguments
        // to the script
        cols := 5
        border := 20
        textSize := border * 0.75

        cWidth := crop at(2)
        cHeight := crop at(3)
        mWidth := ((cWidth + border) * cols) + border
        rows := (nsteps/cols + 0.5) roundDown
        mHeight := (cHeight + border) * rows + border

        outBox := list(0,0, mWidth, mHeight)
        // make a big image to put all the little copies into
        result := IceImage FilledImage(in ComponentType, in Ply, outBox, list(0))

        // crop out the part of the source image we're going to work on
        src := in SubImage(crop)
        e := elow
        for (r, 0, rows-1,
            for (c, 0, cols-1,

                offset := list(border+c*(border+cWidth),border+r*(border+cHeight))
                small := src Move(offset) ExrToneMap(e, 0, 0, 5, 2.2)

                // The DataBox is the area where the image has pixel data
                result := result CopyFrom(small, small DataBox)

                // Create the label
                label := IceImage DrawString(e asString(1,1), textSize, textSize)
                // Use the same position offset which due to where DrawString
                // renders text means this will convienient be in the border
                label := label Move(offset) Shuffle(list(0,0))
                result := result Over(label)

                if (e == ehigh, break)

                e := e + ((ehigh - elow) / nsteps)
            )
        )

        return result
    )
```

You might notice that in this final version we have combined some operations on one line. So the construction of the *small* variable is the result of a *Move*, followed by a *ExrToneMap*.

To put the text onto the montage we used the *Over* operator, which implies the presence of an alpha channel. Since the *DrawString* operator doesn't provide an alpha channel we fake one by duplicating the single channel in the text image using *Shuffle*. *Over* doesn't mind if the background (the montage) is rgb and the text is single channel. It just extends the single channel out so your text will be white.

Here's a sample run made on a image of Mount Tamalpais in Marin County, CA. The source image was one of the sample images you can find at the OpenEXR website



Tips and Tricks

Optimizing Scenes

To get started, let's open the Maya scene, *optimize.ma*.
(Where are the tutorial files?)

1 OPTIMIZING SCENES FOR RENDERING

Armed with a few general rules, it's easy to optimize your scenes for maximum performance. You can realize large performance gains by taking the renderer into consideration early in your 3D pipeline, during modeling and shading. All models are not created equal, and you'll find geometry has a substantial effect on render times. It is common to find that inefficient modeling and/or shading creates unnecessary bottlenecks when it comes to rendering, and when shots are ready to render it's usually too late to fix poor, inefficient models and shaders. By planning in advance, you can avoid squandering valuable render cycles. In this tutorial we'll go through some of the steps that you can use to optimize your scenes for RenderMan.

Make sure you've opened the tutorial file, *optimize.ma*.

2 USE GEOMETRY RENDERMAN LIKES

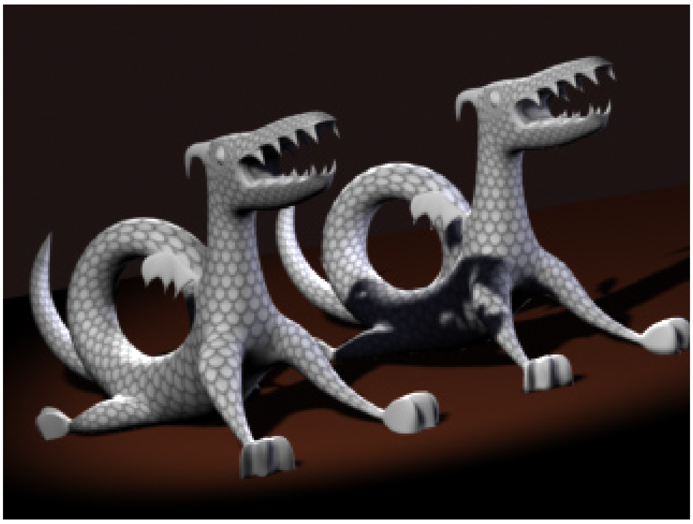
RenderMan prefers NURBS and subdivision surfaces over polygons. In general, NURBS and subdivision surfaces will look better and render faster and more efficiently than an equivalent object modeled from polygons. This holds particularly true for complex curved topologies, where polygons generally require a much denser mesh to approximate curved surfaces.

For simpler models, like a box, the surface type is less important, and polygons may be faster. Generally complex organic objects with lots of curves, like animated characters, are best modeled as subdivision surfaces.

Now Render the Maya scene:

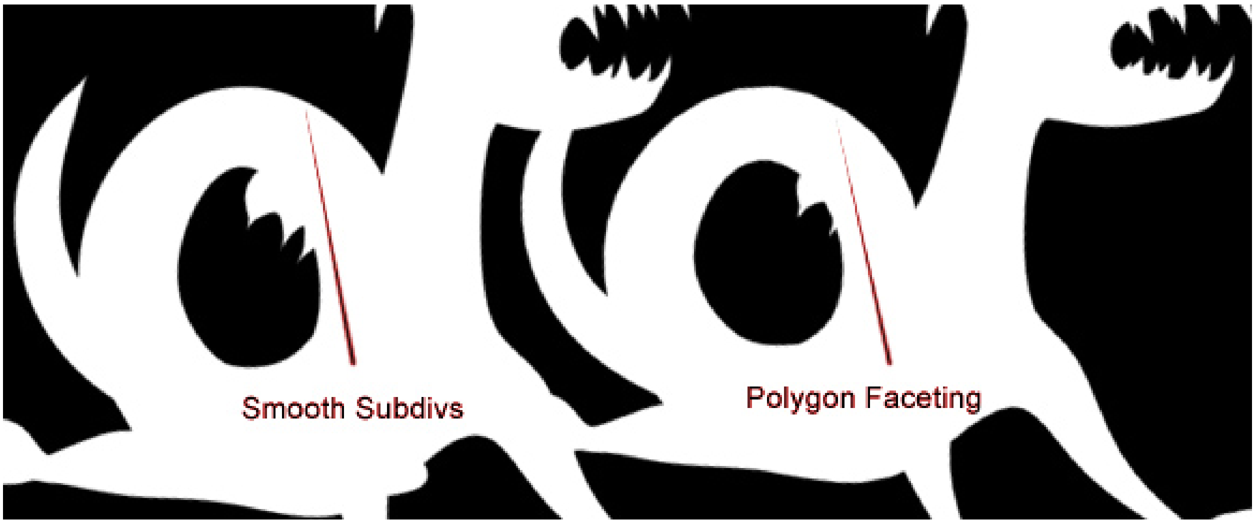
```
Render-> Render Current Frame
```

You should get an image like the one below:



On the left is a subdiv dragon. On the right is a polygon dragon

RenderMan renders true-curved surfaces (tesselating geometry into micropologons on a per-pixel basis) so there are never any tesselation issues with spheres or other curved surfaces. These surfaces will always be smooth, never faceted. If we take a closer look at the alpha image in the scene above, the undesirable faceting of polygons is clear. We can see this in the image below:



The alpha emphasizes the faceting in the poly model

Because RenderMan prefers subdivisions over polygons, go ahead and delete the polygon dragon.

3 USE SINGLE-SIDED SURFACES

Whenever possible, use single-sided geometry. This allows the renderer to throw away half of the surface with no further processing, doubling the rendering speed for that object. Obviously, an object should not be single-sided if the interior or back faces can be seen. Next we'll turn the dragon into a single sided object.

```
1) Select the dragon in Maya.
2) Open the Attribute Editor.
3) Toggle off Render Stats-> Double Sided.
```

The dragon will now be more efficient to render. Setting double-sided surfaces to single-sided will make a big difference when rendering larger scenes.

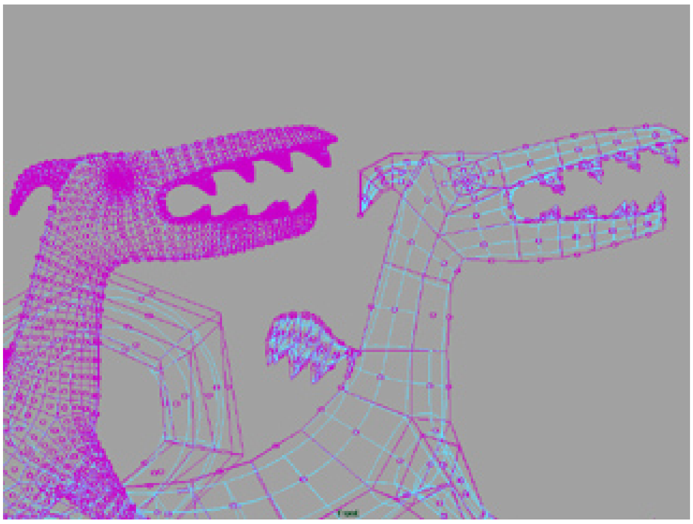
4 AVOID OVER-MODELING

Adding too much detail to a model can slow rendering down. During the modeling phase consider the final placement of an object in a shot, and if a detail is ultimately smaller than a pixel, don't model it. During the modeling stage it can be tempting to build as much detail into a model as possible. Avoid that temptation. Add only the detail that's required and render times will dramatically improve.

Also, avoid unnecessarily dense geometry. Polygon meshes should be just dense enough so that tessellation artifacts are not a problem. Subdivision surfaces should always have as few vertices as possible. NURBS should have only as many isoparms as required. Extra geometry means extra work for the renderer. Take the case of a NURBS sphere it is equally "round" with six or 600 isoparms. In either case it's a sphere, but you will notice that the NURBS sphere with 600 isoparms takes much longer to render.

Ultimately, it's not always feasible to model for each shot, but it is good practice to be aware that the complexity of models has a direct impact on render times.

The subdiv dragon in our scene has the right amount of geometry. In the image below we can compare this dragon (as seen on the right) with another dragon (on the left) which has too much geometry. Both models look the same when rendered, but the heavier model on the left carries a greater, and unnecessary, overhead.



The model on the left is inefficient too much geometry.
The model on the right uses less geometry to define the same form.

5 USE SHADERS TO ADD DETAIL

Sometimes it's better to apply displacement (or bump) shaders to simple geometry than to actually build complex geometry. Take the example of this dragon. Perhaps we want to add scales. These scales could be modeled outright, which would create a much heavier model. Alternatively, a displacement shader can be used to add scales to the dragon much more effciently. By letting RenderMan handle small details during rendering, you'll be able to render lighter models faster. As a bonus, you will also accelerate your interactive Maya sessions because you are using lighter models.

To attach scales to the dragon, we'll attach a pre-made displacement shader.

```
1) Select the shader "scales_lambert" in the Hypershade
2) From the Hyper shader pick:
        Edit-> Select Objects with Materials
3) Now attach the displacement shader. From the Rendering Menu select:
        Lighting/Shading-> Assign Existing Material-> scales_displaced_lambert
```

Now render the scene again:

```
Render-> Render Current Frame
```

You should get an image like this, with nice displaced scales:

A dragon with scales added by a displacement shader

6 SUMMARY

Following a few simple guidelines during modeling and shading will have a positive impact on render times:

1) Use efficient geometry (NURBS and subdivision surfaces) for complex organic models.
2) Use single-sided surfaces and process half as much geometry.
3) Don't over-model. Model for the shot as much as possible.
4) Use shaders to add detail. Let RenderMan do the modeling, and accelerate your renderings and interactive workflow in Maya.

Rendering Efficiently

To get started, let's open the Maya scene, *rendering_efficiently.ma*.
(Where are the tutorial files?)

1 KNOBS AND BUTTONS

There are various controls that you can use to adjust render speed and quality. Often you'll find that you want to use different settings during different stages of the rendering process. For preview rendering, lower quality settings are often an acceptable trade-off because that allows for faster iterative renderings. For final frames, high quality setting are essential. This tutorial will give an overview of which RenderMan quality settings have the biggest impact on your scenes.

Go ahead and open the tutorial file, *rendering_efficiently.ma*.

2 IMAGE RESOLUTION

Image resolution has a substantial impact on the time it takes to render a frame. By rendering smaller images, you can substantially increase the speed of your preview iterations. An image with a resolution of 640x480 will render much faster when the resolution is reduced to 320x240. Often a small test image is all that is necessary to establish the basic brush strokes for a shot. Similarly, half-resolution or quarter-resolution images can be used quite effectively to experiment with large-scale changes to a scene, like shadow placement.

First render the Maya scene at 640x480:

```
Render-> Render Current Frame
```

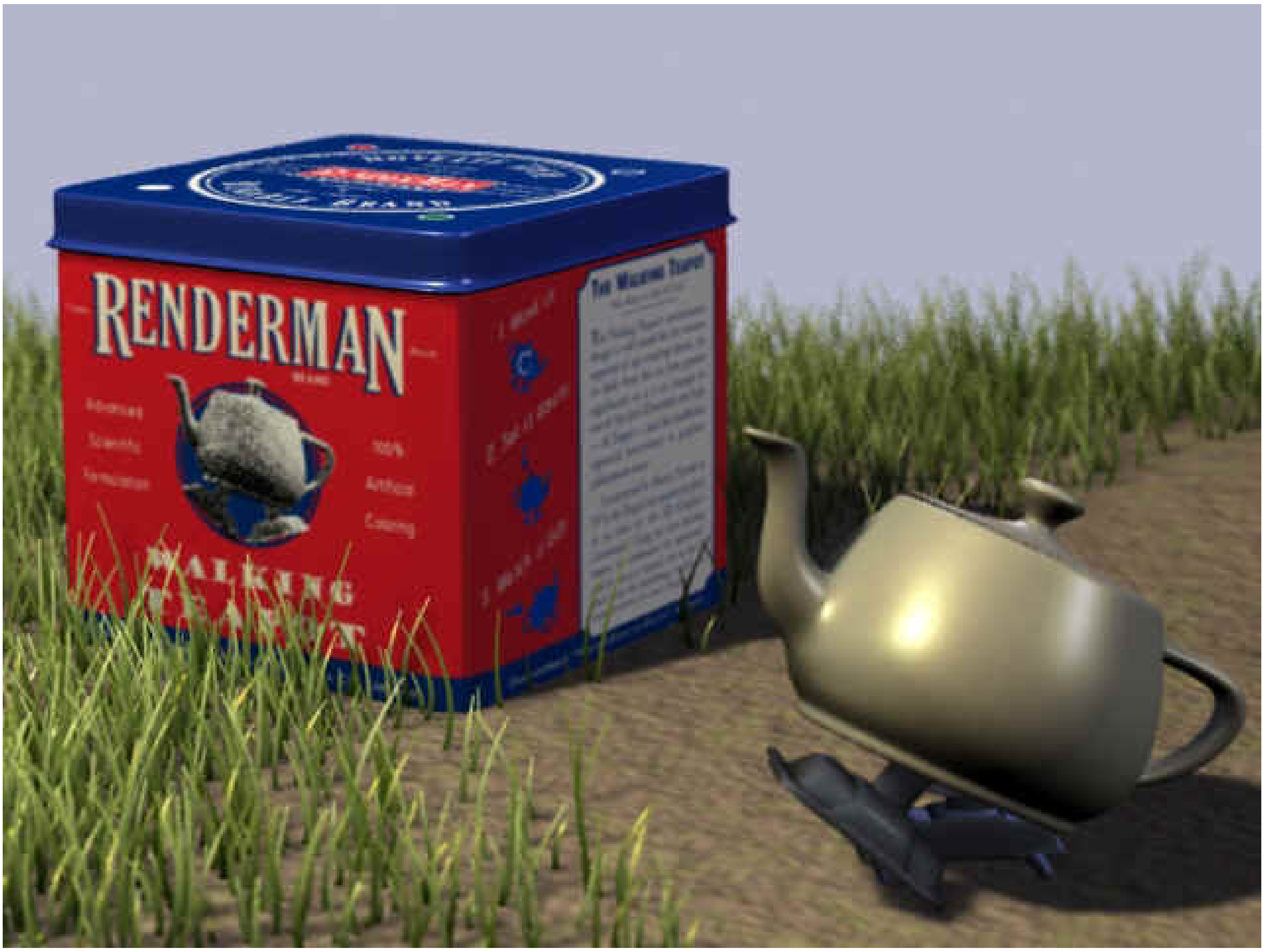


Image rendered at 640x480
(Rendering time: 59 seconds)

Now use Maya's "Test Resolution" Render Globals settings to switch from 640x480 to 320x240 and render the scene again:

```
        Render-> Render Current Frame
```

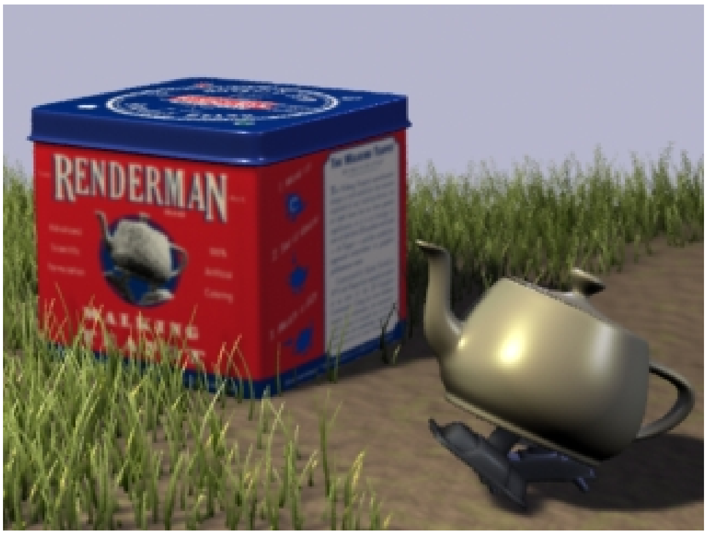
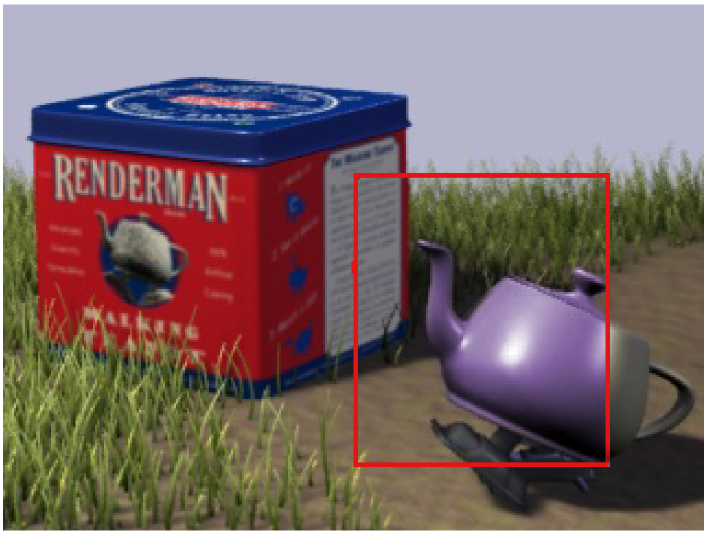


Image rendered at 320x240
(Rendering time: 21 seconds)

By rendering the scene at half the original dimensions the render time is cut by over half.

3 CROP RENDERING

To further accelerate the process of iterative rendering you can render small areas of an image, using Maya's *Render Region*. This allows you to render only the bit of the image you're interested in, speeding up rendering even more. For example, in the image below Render Region is used as the color of the teapot is tweaked.



Render Region can be used to render isolated details (Rendering time: 4 seconds)

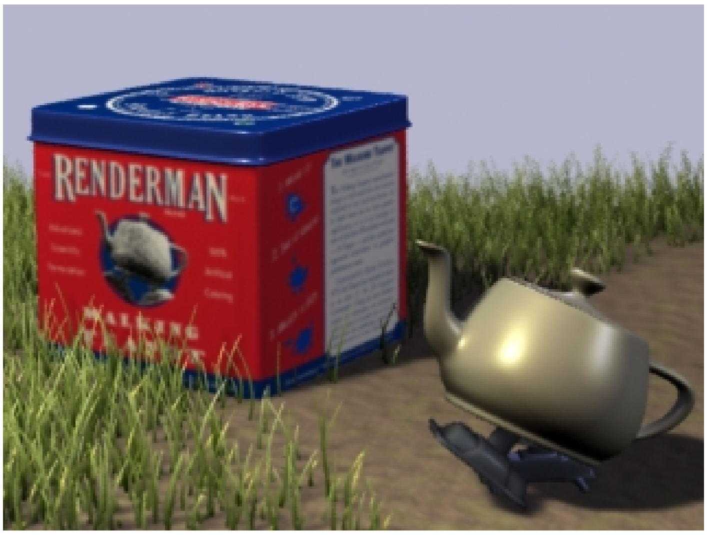
4 SHADING RATE

Shading Rate is the main global quality control for RenderMan. It's also probably the second most critical factor in speed and performance (next to the resolution). A typical shading rate for final rendering is 1.0, or less. Preview renderings can usually be done at 3.0, 5.0, or even 20.0. What is the trade-off with a high shading rate? A shading rate that is too large tends to give blocky looking colors and excessive blur on textures. As long as the color of an object changes slowly and smoothly across its surface, this will look fine. If the surface has drastic color changes, such as sharp-edged patterns in its textures, these results will be unsatisfactory.

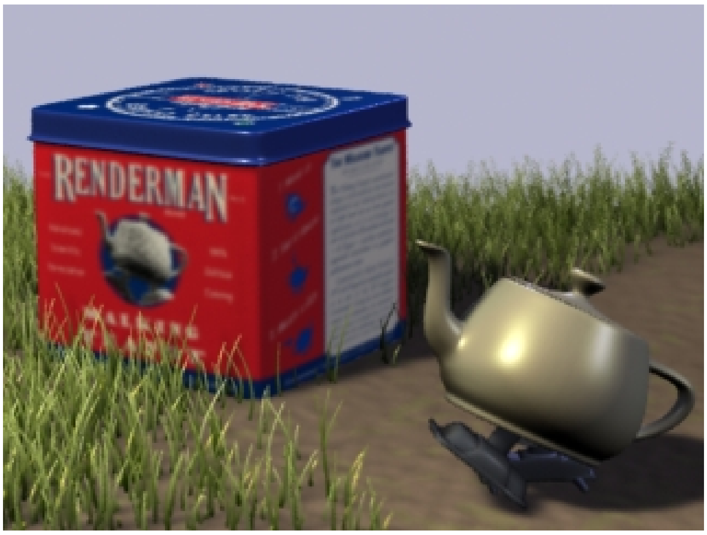
Shading rate controls two important factors. First, shading rate governs the quality of shaders. Second, shading rate governs how many micropolygons are required on a per-pixel basis. A smaller shading rate creates more micropolygons, requiring more memory and overhead. Doubling the shading rate (from 1 to 2) usually gets you nearly twice the rendering speed.

You can adjust the shading rate in the RenderMan Render Globals *Quality* tab. Next render the scene with a shading rate of 2, then 5, and then 10.

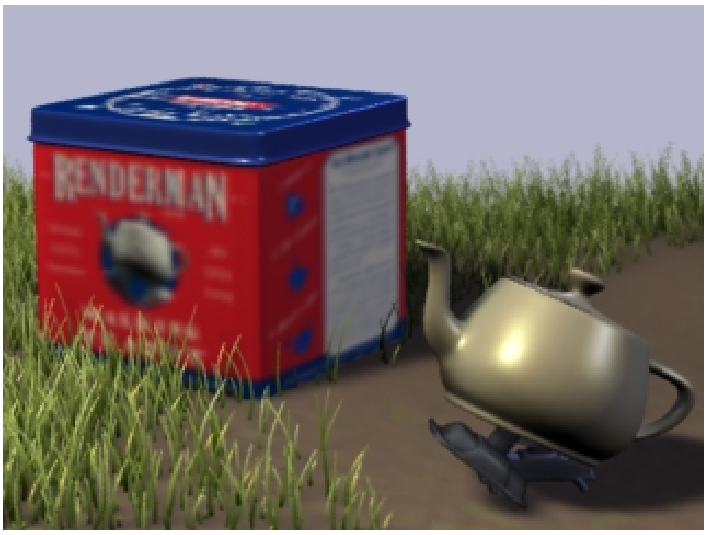
Here's a comparison of different Shading Rates. Notice that, as the shading rate increases, the textures, bump, and shadows get blurrier, until (at a shading rate of 10) the bump is gone, the texture is unreadable, and the shadows are very blurry. Also notice that the shading rate affects shader quality the most, while the geometry (the grass, teapot, and box) retains the original form:



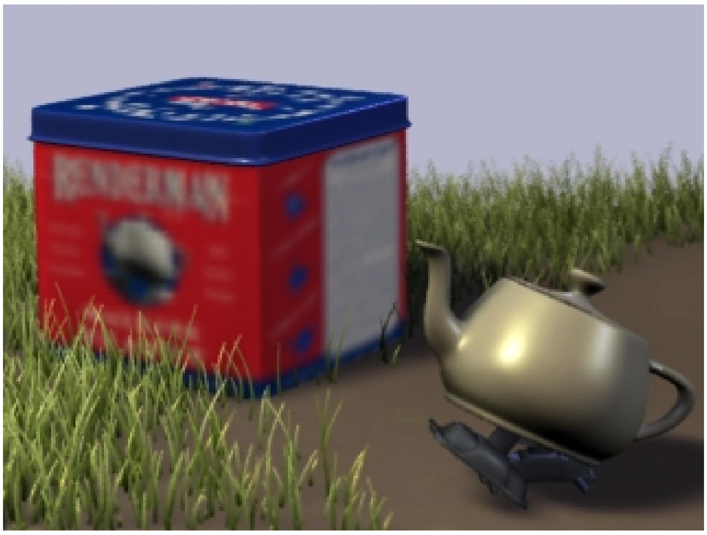
Shading Rate of "1"
(Rendering time: 21 seconds)



Shading Rate of "2"
(Rendering time: 14 seconds)



Shading Rate of "5"
(Rendering time: 9 seconds)



Shading Rate of "10"
(Rendering time: 8 seconds)

One important quality of shading rate is that it can be set globally, for the entire scene, or it can be changed from one primitive to the next. This allows shading rate to be set at a higher quality setting for only those objects that require it. For example, in this scene the teapot has a very simple shader, while all the other objects have much more detail. We could speed up our rendering (with no undesired artifacts) by attaching per-object shading rate to the teapot. To create a per-object shading rate, simply add the RenderMan shading rate attribute to the teapot. To attach the attributes, follow these steps:

1. Select the teapot
2. Open the object in the Attribute Editor.
3. From the Attribute Editor Menu:
   `Attributes-> RenderMan-> Manage Attributes`
4. Select "Shading Rate" from the menu and click "Add"
5. Set the new Shading Rate attribute to "5.0" It will appear in the "Extra RenderMan Attributes" sub-tab.

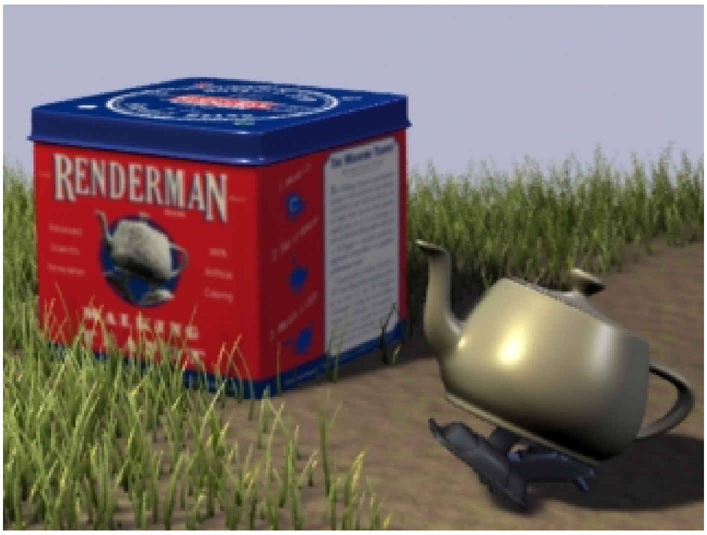
Per-object shading rates are a very powerful technique that can be used optimize rendering.

5 PIXEL SAMPLES

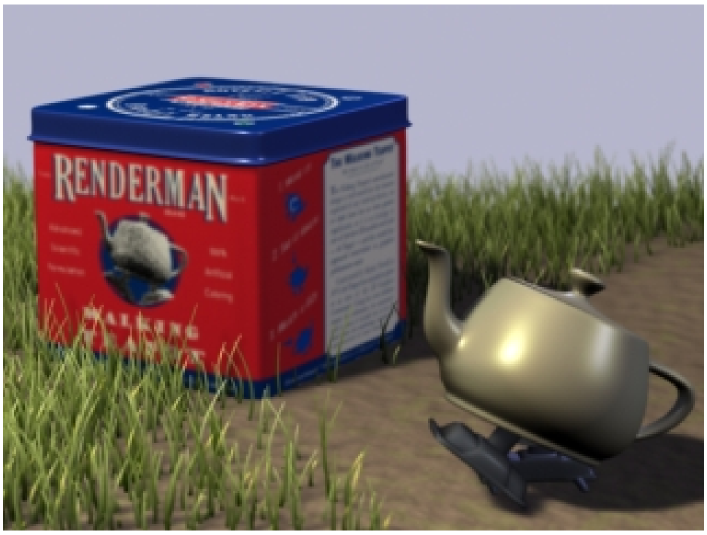
RenderMan provides advanced antialiasing controls, and Pixel Sampling is used to adjust the amount of super-sampling around a pixel. A setting of 3x3 is a medium quality setting and may be sufficient in many cases. You'll find, however, when rendering scenes with fine detail, like fur, or motion blurred objects, that higher values of 4x4, 6x6, or even 11x11 are required to produce an artifact-free image.

In this scene the depth of field effect is grainy (you can see the grain in the grass on the horizon). To create a smooth depth of field effect, increase the pixel samples, which can be found in the RenderMan Globals *Quality* tab. Set the pixel samples to 11x11. Render again.

You should get an image like the one below, with a nice smooth depth of field effect:



Pixel Samples of "3 x 3"
(Rendering time: 21 seconds)



Pixel Samples of "11 x 11"
(Rendering time: 32 seconds)

6 RECOMPUTE SHADOWS (AND OTHER PRE-PASSES)

When tweaking shaders, an image is often rendered over and over again. It most efficient, of course, to do as little redundant work as possible. One shortcut is to reuse shadow maps (and other pre-passes) as opposed to calculating them every frame. This technique can dramatically increase the speed of renders that use multiple large shadow maps. Other pre-passes can be cached in much the same way, allowing the reuse of occlusion, subsurface scattering, and other effects. To reuse the shadow map in this scene, go to the Shadows settings in the Attribute Editor for the `KeyLightShape` and toggle the *Disk Based Dmaps* setting from Off to Reuse Existing Dmap [s] .

Now render again. The shadow map will be reused and will not be computed each time you render.

Recomputed shadows

Certain other passes, such as GlobalDiffuse, created for Global Illumination, have their caching behavior controlled via the Passes Tab of the Render Globals.

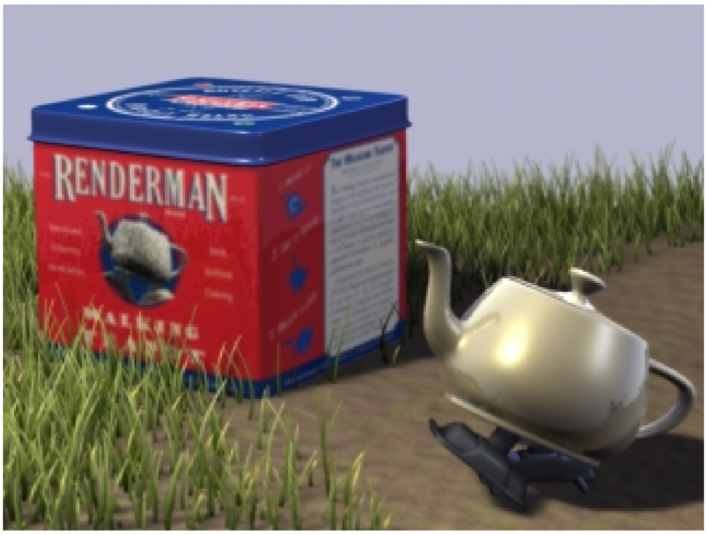
1. In the *Passes* field select the pull-down menu (from the downward pointing black arrow) and select the pass.
2. Open the *Settings* sub-tab.
3. Switch the *Caching Behavior* from *Compute* to *Reuse*.

7 ENABLE RAY TRACING ONLY WHEN REQUIRED

In RenderMan ray tracing is implemented as a hybrid sub-system, which means ray tracing must be enabled to get effects such as ray-traced reflections, refractions, and shadows. Ray tracing is, by nature, much less memory efficient than pure scanline rendering. Your scenes will render faster and more efficiently when ray tracing is disabled.

In this case, we can get accurate reflections from the teapot and the box by ray tracing the scene. Enable ray tracing by opening the *Features* tab of the RenderMan Globals and clicking the ray tracing box. Now render the scene again.

The image will take longer, but it will have accurate reflections:



Ray traced reflections
(Rendering time: 2 minutes)

8 SUMMARY

You can tune your renders to get the most out of RenderMan if you know which buttons to push. For preview renders, use the lowest quality settings that are acceptable. For final images, increase the settings for high quality images, but take care to only increase the settings as much as is required to avoid extra work. If you familiarize yourself with these speed-quality knobs and buttons, you'll get more renders through production:

1. Use the lowest image resolution as possible, particularly during preview rendering.
2. Take advantage of crop rendering when tweaking individual details.
3. Choose an appropriate shading rate during preview and final images. The appropriate shading rate is the largest value which gives you an acceptable image. Remember to take advantage of the ability to attach individual shading rates to objects.
4. Adjust pixel samples to accerlate your renders. Use the lowest quality values for pixel samples as possible, but for final high quality images, pixel samples may have to be increased as high as 12x12 to eliminate aliasing artifacts. Again, use the mininum amount of samples that achieves a high quality image.
5. Since ray tracing carries substantial overhead, only enable ray tracing when required.

Pixar Animation Studios

Recipes

RENDERMAN RECIPES

The RenderMan Recipes are a collection of "canned" Maya files with effects already set up. Recipes are not full blown tutorials, just simple scenes containing a variety of effects made using RenderMan for Maya. By examing these scene files, it's possible to see how these effects have been created.

Pixar Animation Studios

Prev | Next

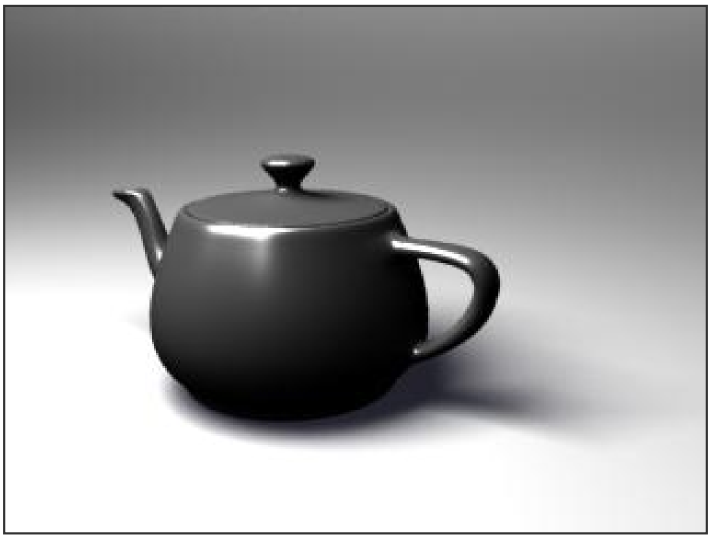
Scenes and Shaders

To get started, you'll want to open the appropriate file for the recipes.
(Where are the Maya files?)

RECIPE #1: AREA LIGHT SHADOW

Open the scene, /scenes/*recipes/area_light.ma*.

In this scene an area light is set up to cast soft shadows. The shadows are ray traced and the area light has RenderMan Attributes attached to it to control how many ray samples the area light casts. More samples provide a higher quality shadow. This scene uses 256 samples to cast an artifact-free shadow.



Area light with 256 ray traced shadow samples

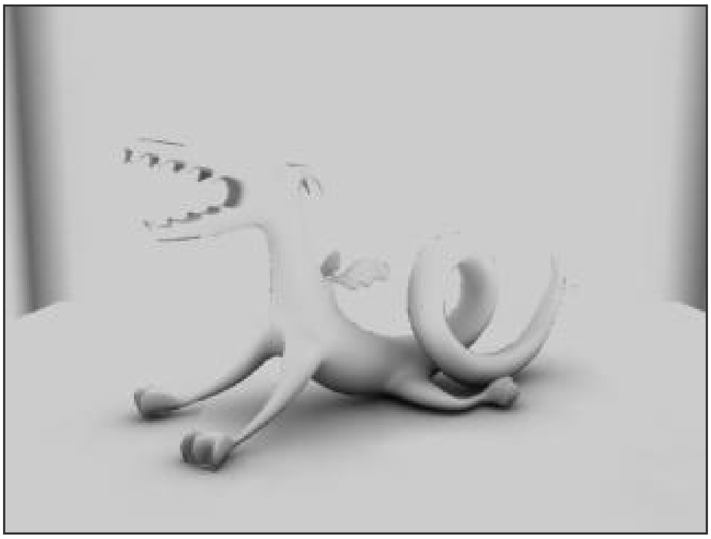
RECIPE #2: INTERIOR OCCLUSION

Open the scene, /scenes/*recipes/interior_occlusion.ma*.

In this scene we have our dragon in a box, with a simple Lambert attached to all the surfaces. The lighting is courtesy of a RenderMan Environment Light node. Straight out of the box, the defaults have the Max Dist parameter set to 10000, which, as you can see, doesn't get the job done:

Max Dist = 10000

For global illumination to work properly with interior scenes, you need to shorten the maximum distance that rays are shot to compute occlusion, meaning that you need to stop well short of your ceiling, or your scene will be fully occluded. In general, a good guideline for setting the Max Dist parameter is one-tenth the distance from your ground plane to your ceiling. In this scene, the ceiling is about 20 units from the ground plane, so we have set the Max Dist to 2.0, and, as you can see, we get a much more useful result:

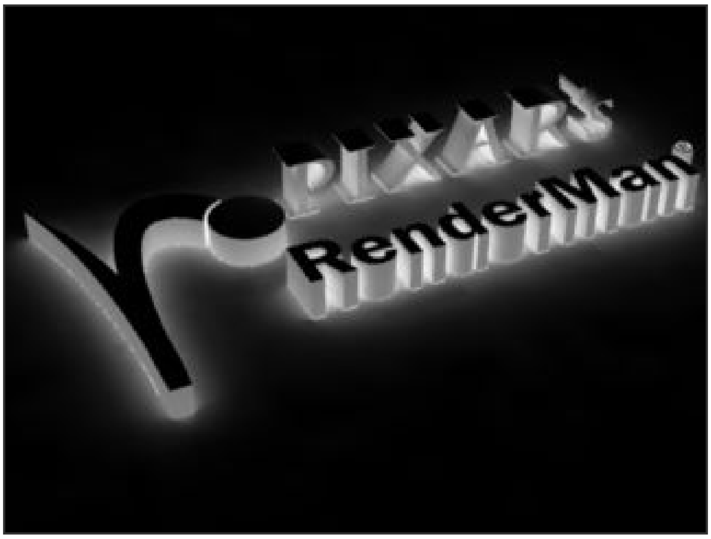


Max Dist = 2.0

RECIPE #3: PURE OCCLUSION

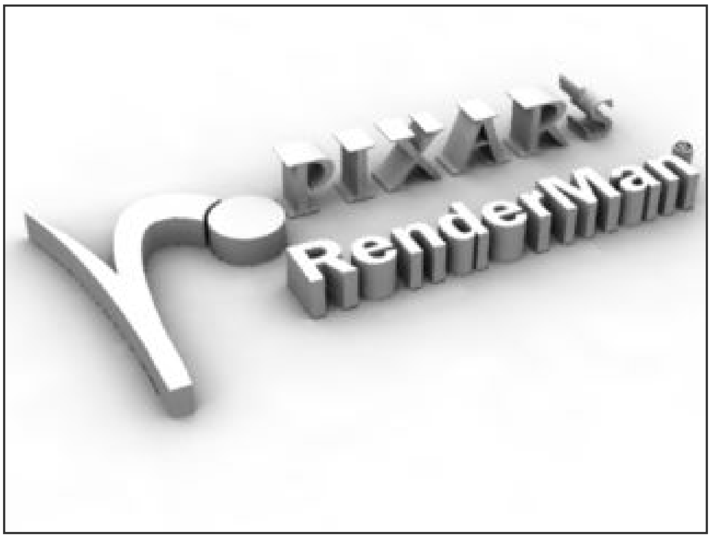
Open the scene, /scenes/*recipes/pure_occlusion.ma*.

This scene outlines two ways to get a pure occlusion image, as seen in theGlobal Illumination tutorial. The correct way to get this so-called pure occlusion is to create an additional, secondary output for occlusion, via the Passes tab of the Render Globals, as we have done here. This output, however, is the actual occlusion data, as read and used by the renderer, and is the inverse of what you might expect to see:



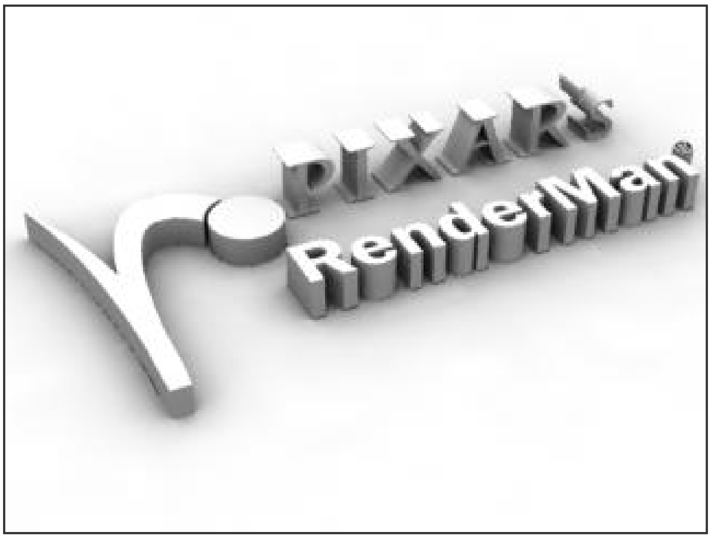
Actual pure occlusion

You can, however, then invert that image with the tool of your choice, such as fCheck, which would give you this image, suitable for compositing:



Inverted occlusion

Or, you can fake it by using a simple Lambert shader with the color set to white and the diffuse value cranked to 1, giving you the following image in your Final rendered image:



Faked pure occlusion

Prev | Next

Prometheus   Rendering a Greek God

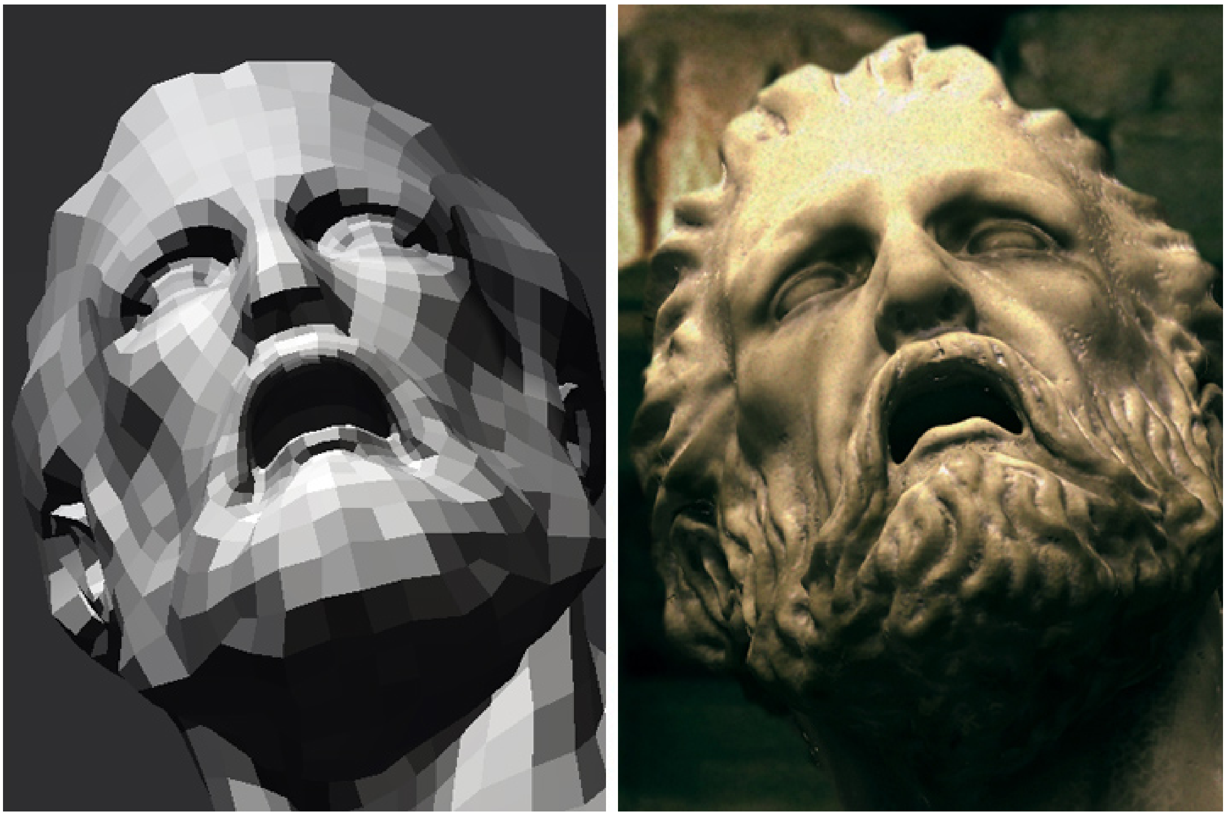
The Prometheus tutorial was created by guest author, Scott Eaton, of Escape Studios in London.

In the Prometheus tutorial, we will use RenderMan for Maya to shade, light and integrate a 3D model into a background plate from the British Museum. During the process, we will explore many of the advanced features that RenderMan for Maya offers including: micropolygon displacements, subsurface scattering, ambient occlusion, brickmaps, and secondary outputs. The tutorial will be broken into three parts:

First, we will develop of a marble shader that approximates the main surface characteristics our reference images. It will need to show small divots from the wear of time on the surface of the marble, discoloration from oxidation, and changes in specularity from polished areas to rough, unfinished areas. And, of course, no marble shader would be complete without subsurface scattering. As the shading network builds up, refer to the Maya materials in the scene files. Each important node will have notes on it to explain its function in the shading network.

Second, we will analyze and match the lighting setup in the British Museum. Without the benefit of an HDRI probe to light from directly, we will have to dissect the lighting from reference photos and an analysis of how sculptures are lit in the British Museum.

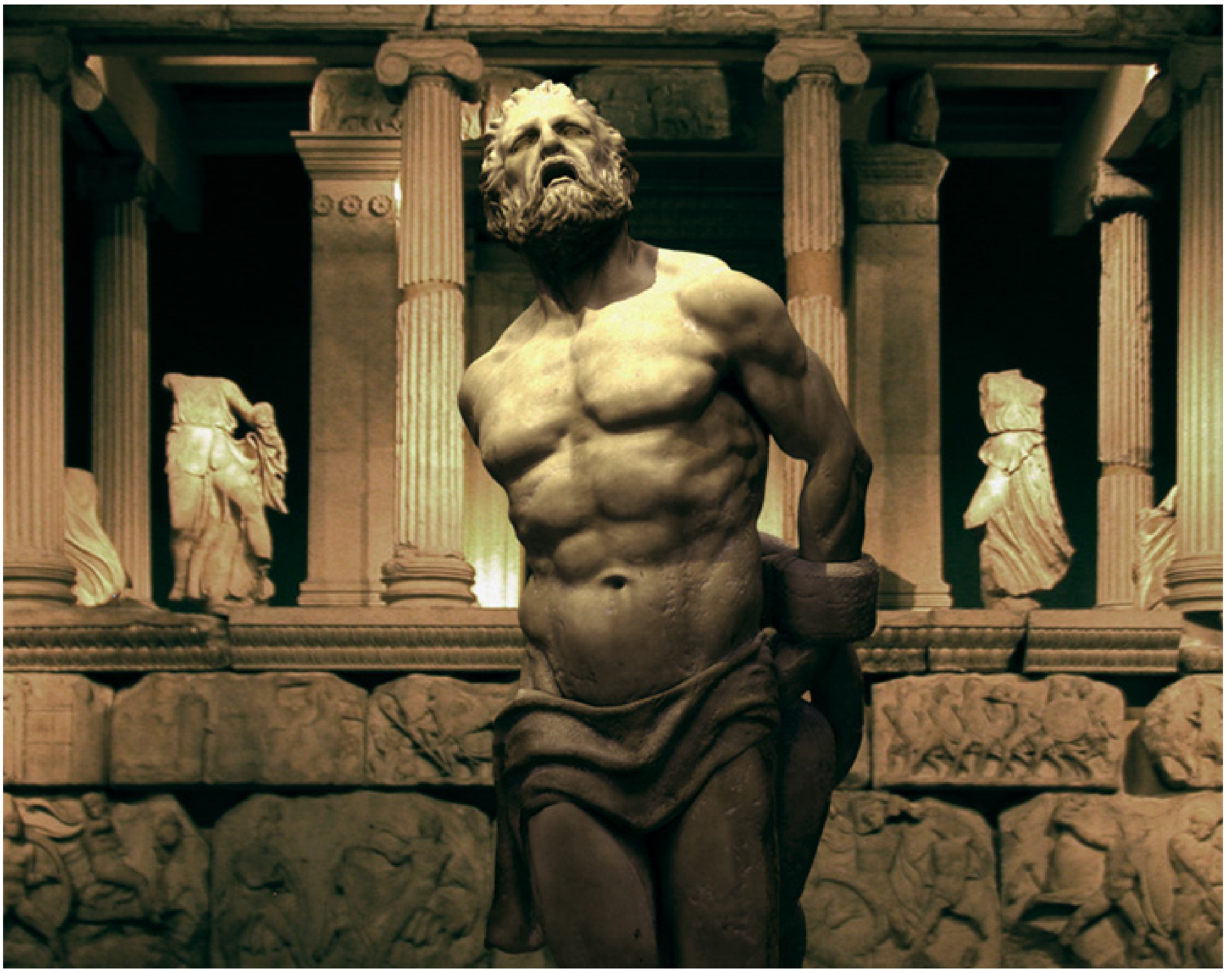
Third, we will render out a number of secondary outputs (diffuse, specular, occlusion, etc) with RenderMan for Maya, and use these secondary outputs to build up a composite over a background plate from the museum. This step is where the magic and flexibility of secondary outputs pays off as we layer and correct each output to match the museum plate.



Before and After - In this tutorial we'll add complexity to a simple model using

displacements, occlusion, subsurface scattering, and secondary outputs

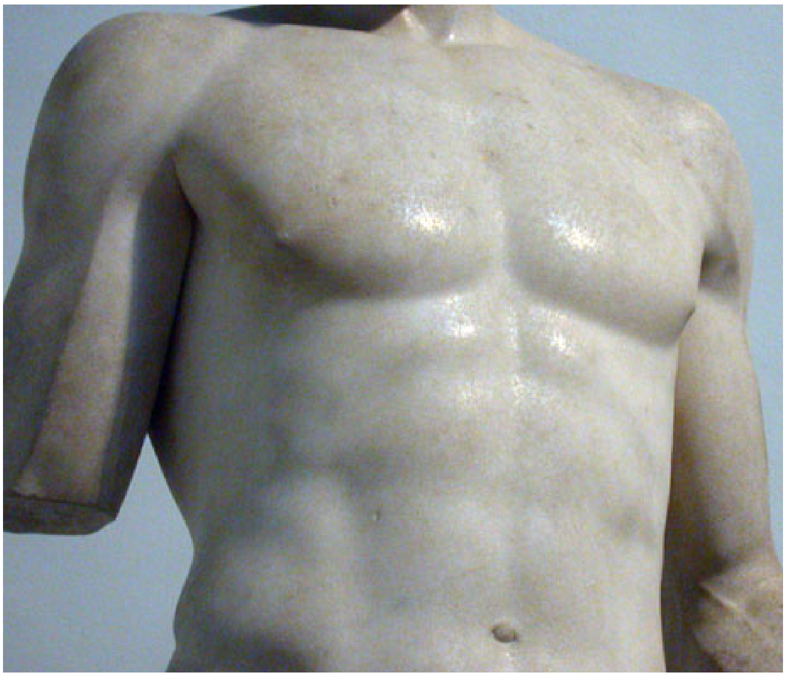
This is an advanced tutorial combining a number of different workflows, and because of that, this tutorial requires a good understanding of RenderMan for Maya. You may need to review certain techniques elsewhere. A key strategy of this tutorial is the use of Secondary Outputs, which will allow us to use 2D compositing to finalize our lighting. At the end of the tutorial, the final result should look something like this:



The end result, the 3D model of Prometheus composited over the background image.

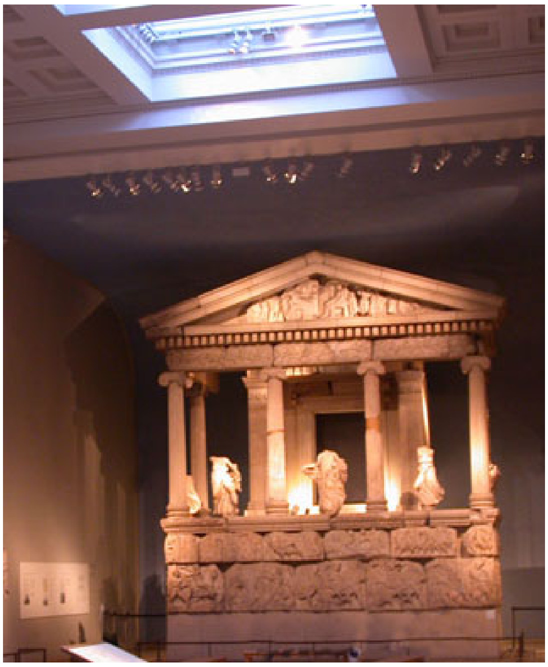
1 ANALYZING REFERENCE MATERIAL

Before we get started, open Maya and make sure that RenderMan for Maya is properly set up.

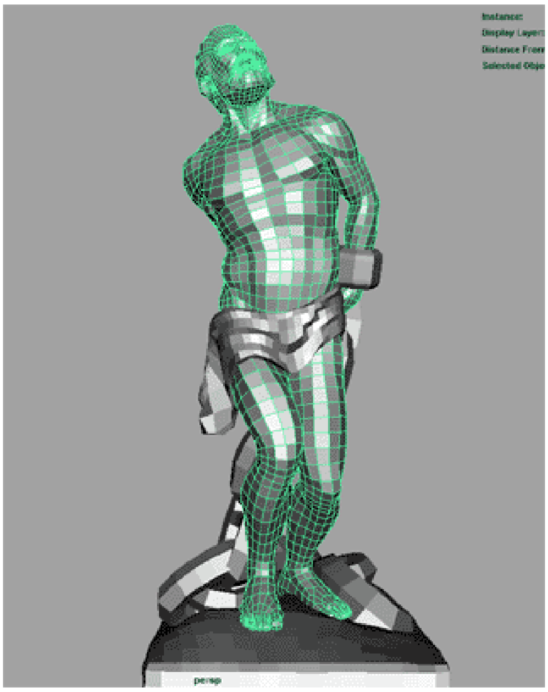


Weathered marble shader reference

1. The surface we are trying to mimic is aged, weathered marble. In our reference image above, we can see that marble is not simply a 3D marble texture mapped onto a Blinn material. The surface is actually a layered combination of cloudy-grey base texture, yellowish surface discoloration from environmental wear, black specs from mineral contamination, faint rain streaks, and whitish marble dust build-up in the deep cavities. Creating a convincing marble will be a challenge.



2. The lighting in the gallery is made up of both direct and indirect light sources. In our reference photo above we can attribute a large amount of non-directional diffuse light to the skylight above the gallery. In addition, each sculpture is individually lit with warm, colored spot lights located near the ceiling.



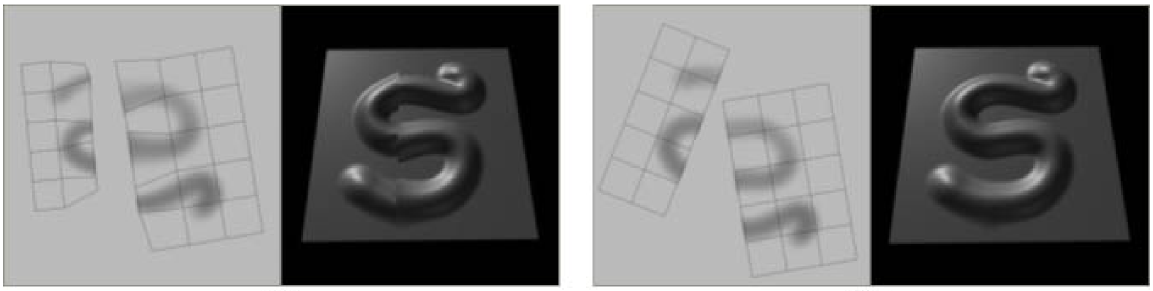
3. Open the Maya tutorial scene, ***Stage1.mb**,* to get started on the tutorial. (Where are the tutorial files?) The initial scene contains a simple light setup and the three parts of our statue: *Prometheus*, *cloth*, and *chains*. Our first step is to add comlexity to this model using displacement maps.

2 SHADING: SUBDIVISION AND DISPLACEMENT

4. We will begin by adding displacement maps to the 3D model which will add the intricacies of a carved marble statue to our lightweight 3D model. Luckily for us, we already have displacement maps for this purpose, originally created using ZBrush. Because it is capable of producing high quality displacement maps, ZBrush is a useful tool for use with RenderMan because RenderMan is capable of rendering these displacement maps extremely efficiently.

●Tip

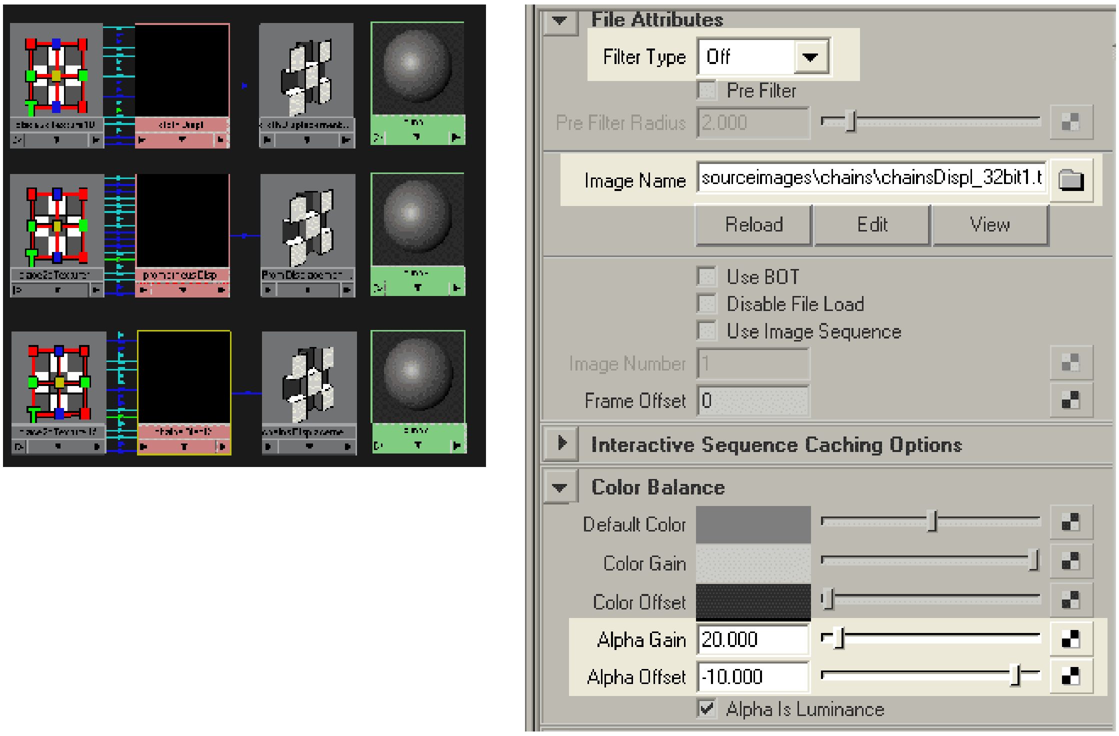
Watch UV Seams: When rendering displacement maps it is important to create equal UV seams, otherwise artifacts may appear in the displacement along the UV seams. In the images below, the image on the left has unequal seams, while the image on the right has equal seams.



No

Yes

For the highest quality displacements, we'll want to render the model as a subdivision surface. In RFM, this is as easy as adding a single attribute to the shape node. Select each piece of the statue, one piece at a time, and in the attribute editor select: *Attributes-> RenderMan-> Add Subdivision Scheme*. Do a test render to confirm that you have a perfectly smooth subdivision surface.



File Attributes

Filter Type: Off

□ Pre Filter

Pre Filter Radius: 2.000

Image Name: sourceimages\chains\chainsDispl_32bit1.t

□ Use BOT

□ Disable File Load

□ Use Image Sequence

Image Number: 1

Frame Offset: 0

Interactive Sequence Caching Options

Color Balance

Default Color

Color Gain

Color Offset

Alpha Gain: 20.000

Alpha Offset: -10.000

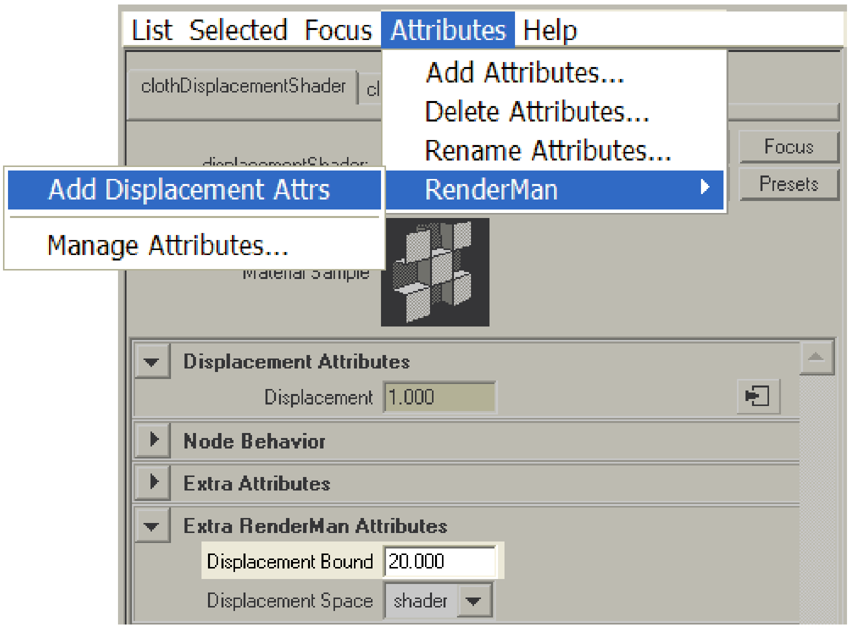
☑ Alpha Is Luminance

Three Blinn materials with displacements are added to the three parts of the statue

5. Connect three standard Blinn materials to each of the three pieces of the statue, and assign a file texture as a displacement map, as shown above. Get the 32bit displacement maps for each model from the source images directory, *sourceimages-> prometheus_tutorial-> [image folders]*. In each file node set the *Alpha Gain* to 20, *Alpha Offset* to -10. This centers the displacement on 0. A final task: on the file node, turn *Filter Type* to off. This stops filtering on the displacement map, which can reveal seams.

●**Tip**

Displacement mapping: RenderMan for Maya brings RenderMan's extremely fast and accurate displacement mapping to Maya users. Getting used to having all this power takes time and a shift in thinking about geometric detail. Because RenderMan renders displacement mapping as quickly as bump mapping (nearly), there is no reason not to use it extensively in your scenes where geometric detail is required. Think, for example, tire treads on a large truck, a stone wall, etc. All can be represented in your scene with lightweight models but displaced at render time for maximum detail. Experiment by mapping a few procedural textures to a displacement node.



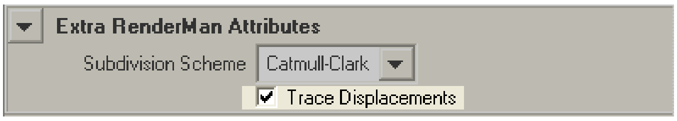
Displacement 1.000

Displacement Bound 20.000

Displacement Space shader

Displacement Attributes are added to the shader and Displacement Bound is set to "20"

6. It is important to tell RenderMan how much you ever intend to displace your surface. This helps the renderer to efficiently process the geometry for rendering. Select each of your displacement nodes and go to *Attributes-> Renderman-> Add Displacement Attrs,* as shown above. To estimate your displacement bound, roughly use your alpha gain from your displacement map node, in this case "20".



Subdivision Scheme Catmull-Clark

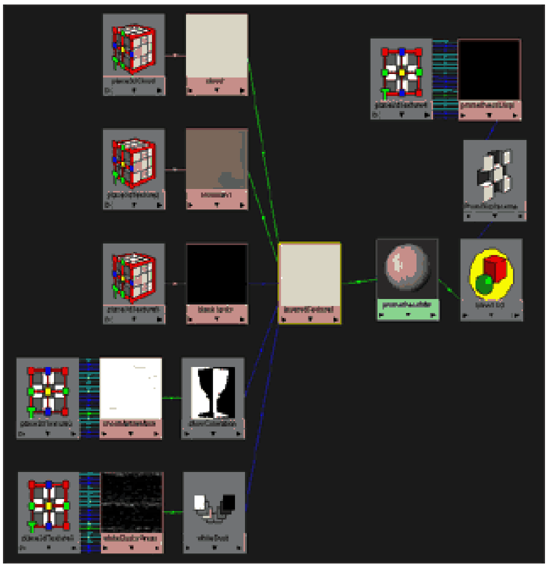
☑ Trace Displacements

"Trace Displacements" are enabled

7. We must also tell RenderMan that we want to use the displacement mapped version of this object for any ray tracing that we are going to do (occlusion, blurred reflections, etc). To do this we must add another RenderMan attribute to the shape node of our models. Individually select each shape node, of the three parts of the statue, and go to *Attributes->RenderMan->Manage Attributes*, then select and add *Trace Displacements*.

●Tip

Quality Control: A single parameter, Shading Rate, controls the quality of shading and displacement sampling in Renderman for Maya. The control is located in the *Render Settings-> Quality* tab. The control is simple but powerful. The value is expressed in pixels and equates to the pixel space between shading samples (in screen space) at render time. Therefore, the lower the number, the higher the quality. A value of one, for example, samples textures and procedural networks once for every pixel in the image; similarly a value of five samples every fifth pixel, and is quicker, but lower quality. Consider raising the shading rate to 5-10 to speed your preview renders, but drop it back down to 1.0 or 0.5 for final renders.

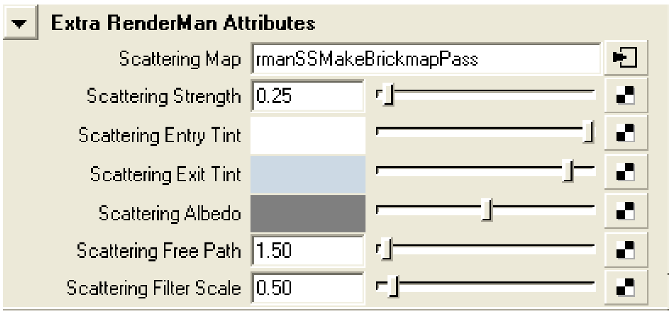


The network for the marble shader in the Maya file, Stage2.mb

8. Next, we can build up the patterns from our reference with a combination of procedural textures, high-level control maps, and texture maps. Now we'll take a shortcut and open ***Stage2.mb*** and investigate how the marble shader is constructed. Notice that the diffuse, specular, and bump components are built up using layers, many of which are shared between shading components. We will use this network as the starting point for adding RenderMan specific features to our materials.

●Tip

Shading in Layers: It is good practice to break your shading up into modular layers. If you are building up a diffuse color (either procedurally or by painting maps), build it in components, starting from the largest "low frequency" patterns and work your way to smallest "high frequency" details. By separating these patterns into layers, it makes it easy to change or modify any one component independently. You will also find that some of these patterns will individually effect the specularity, reflectivity, or bump of the material and can be easily reused there. You will see extensive use of Layered Texture nodes in this shading network provided.



Extra RenderMan Attributes

Scattering Map rmanSSMakeBrickmapPass

Scattering Strength 0.25

Scattering Entry Tint

Scattering Exit Tint

Scattering Albedo

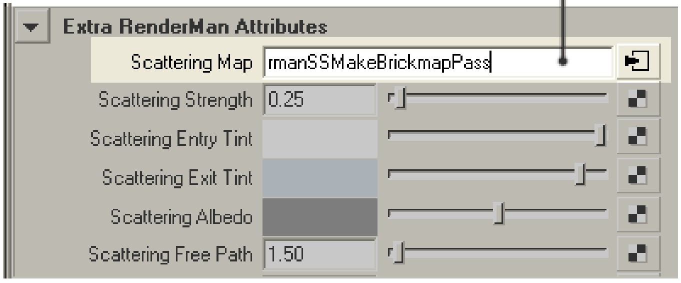
Scattering Free Path 1.50

Scattering Filter Scale 0.50

Add subsurface scattering attributes to the Blinn materials and adjust as shown

9. Next add RenderMan subsurface scattering attributes to our three Blinn shaders. Select each Blinn material and go to the attribute editor and add *RenderMan Attributes-> Subsurface Scattering*. This adds a collection of extra attributes to the shader as well as an additional render pass for creating the SSS brickmap (which also shows up in the *Render Settings-> Passes* tab).

Right click



Extra RenderMan Attributes

Scattering Map rmanSSMakeBrickmapPass

Scattering Strength 0.25

Scattering Entry Tint

Scattering Exit Tint

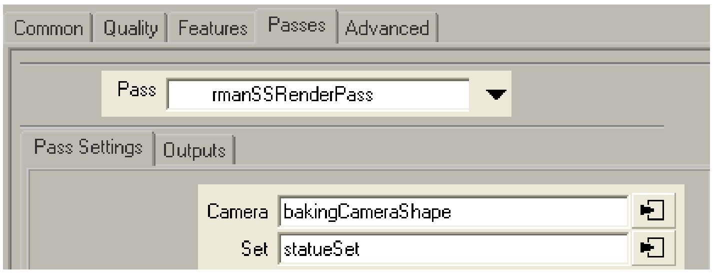
Scattering Albedo

Scattering Free Path 1.50

Set the three Blinn's to share the same Brickmap Pass

10. For the SSS pass we want to include all three pieces of the statue into one pass in order to share

the brickmap between the three Blinn shaders. In each Blinn, right click over the *Scattering Map* field and select the first rmanSSMakeBrickmapPass (delete the other SSS passes from the *RenderSettings-> Passes* tab). Now all three pieces of geometry will be written to and read from one brickmap.



Pass rmanSSRenderPass

Camera bakingCameraShape

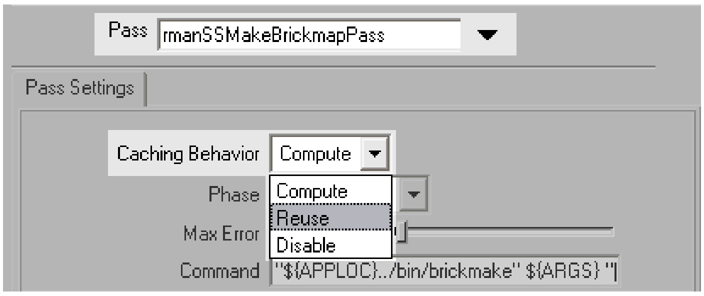
Set statueSet

Configure the camera and set for the SSS pass

11. In *Render Settings-> Passes*, select the *rmanSSRenderPass*. To pick the baking camera, right-click in the *Camera* field and select *bakingCameraShape,* which has already been set up in the scene. To tell the pass which objects to render in the SSS pass, right-click in the *Sets* field and choose *statueSet*. (Note: for best SSS baking results, either enable raytraced shadows on lights or have shadow maps being reused, otherwise your SSS pass will not self shadow). Now you're ready to render the effect. Remember that once the SSS effect is rendered, the SSS brickmap can be reused. Since this statue is stationary and the lights are fixed, we'll be able to reuse the SSS brickmap over and over. Next render the effect then switch the *Caching Behavior* of the SSS brickpass from Compute to Reuse, as shown in the following tip.

●Tip

Reusing Brickmaps: Bake early, bake often. RenderMan uses an elegant baking system to capture computationally expensive effects into 3D caches called brickmaps. It is important understand that baking is done as a separate render pass, and to get best results you should specify a reference camera in the Render Pass settings. By changing the *Caching Behavior* of the pass you are able to explicitly control the reuse of the SSS brickmap.



Pass rmanSSMakeBrickmapPass

Caching Behavior Compute

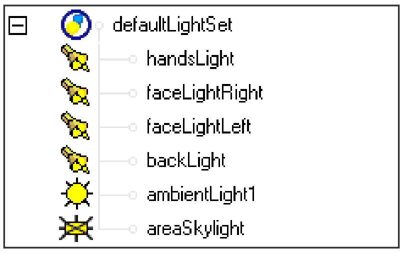
Phase

Max Error

Command "${APPLOC}../bin/brickmake" ${ARGS} "|

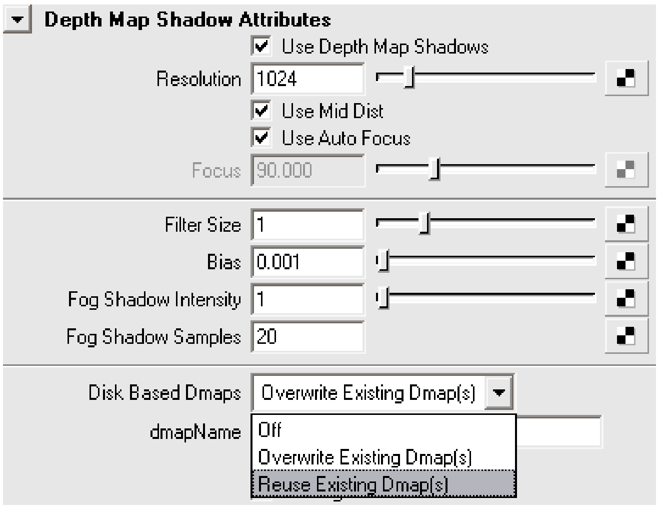
Caching Behavior can be set to Compute, Reuse, or Disable

4 LIGHTING SETUP



Add shadows to the spotlights

12. With the shaders finished, we can now focus on lighting. The scene has two main lighting elements, the spot lights and the sky light. To start, we'll tackle the spot lights. There are four yellowish spotlights that light the sculpture from above. On each of these spot lights, enable depth map shadows.



Depth Map Shadow Attributes

☑ Use Depth Map Shadows

Resolution 1024

☑ Use Mid Dist

☑ Use Auto Focus

Focus 90.000

Filter Size 1

Bias 0.001

Fog Shadow Intensity 1

Fog Shadow Samples 20

Disk Based Dmaps Overwrite Existing Dmap(s)

dmapName

The shadow pass

13. Next we'll render the scene once to generate all our shadow maps, and then we'll set our spot lights to reuse the shadow maps, so that they are not recomputed every time we render. To reuse shadows, change the Disk Based Dmaps setting on each spot light from *Overwrite existing Dmaps(s)* to *Reuse existing Dmaps(s).*

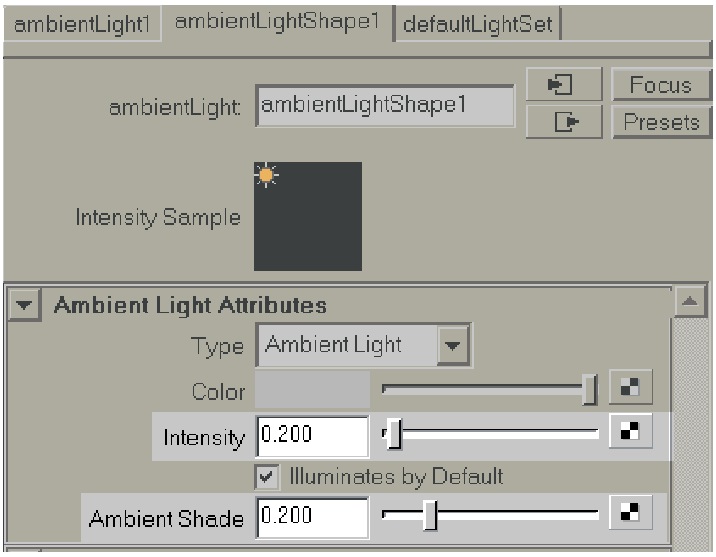
●**Tip**

Map Reuse: RenderMan has a long history of reusing pre-computed maps to save on render time for large productions. Each time you create a new pass that needs to compute a map (be it a shadow map, a reference map, or 3d brick map) you get a *Phase* option to create a disk cache in two ways: *once per job* and *every frame*. For stationary, non deforming geometry (as we have in this tutorial) *once per job* is the option you want to select. For animation sequences where there is motion that will change the map, use *every frame* and you will get a cached map for each frame of the animation.

---

14. To give us control over the shadow filtering and samples once we have a shadow mapped baked, we need to add additional custom attributes to our light shape node. Select the light and go to *Attributes->Renderman-> Shadow Attributes*. This gives us access to the shadow filtering and shadow sampling attributes to blur the edges of our shadow maps as necessary. Now the spot lights are properly set up.

15. Next we'll approximate the skylight in the British Museum. To achieve this, we will combine two effects: an area light for specular highlights and a custom occlusion pass (see next step) for soft shadowing. The area light will be used only to cast accurate specular reflections while the custom occlusion pass will allow us to bake expensive soft shadow calculations. Find the *areaSkylight* in the scene, unhide it, and enable *Emit Specular* only.



ambientLight: ambientLightShape1

Intensity Sample

Ambient Light Attributes

Type Ambient Light

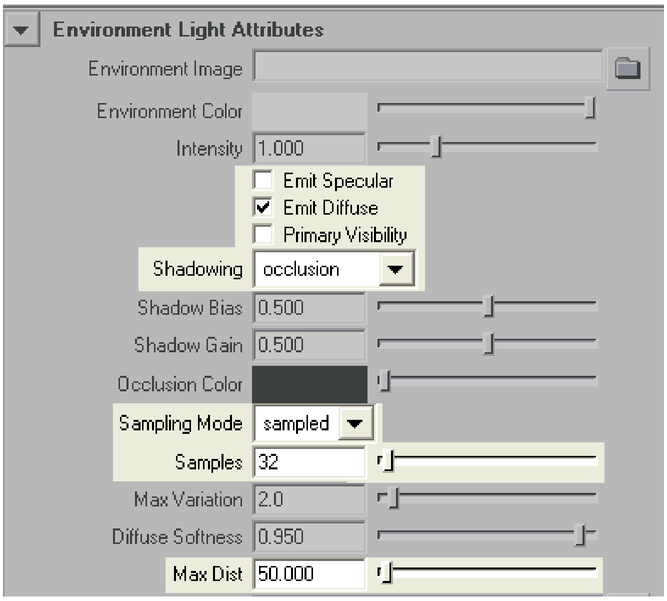
Color

Intensity 0.200

☑ Illuminates by Default

Ambient Shade 0.200

16. Additionally, in the museum there is light reflected from the floor which is cast from the overhead skylight. To create that effect, unhide the ambient light in the scene (we will later use the light to attenuate our ambient occlusion pass). Set your intensity low and tune *Ambient Shade* to 0.2 to give a little direction to our light, to simulate reflection from the ground. The amount of ambient can be tuned later in compositing.



Environment Light Attributes

Environment Image

Environment Color

Intensity 1.000

☐ Emit Specular

☑ Emit Diffuse

☐ Primary Visibility

Shadowing occlusion

Shadow Bias 0.500

Shadow Gain 0.500

Occlusion Color

Sampling Mode sampled

Samples 32

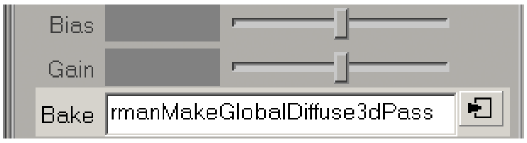
Max Variation 2.0

Diffuse Softness 0.950

Max Dist 50.000

Set up your environment light as shown

17. To create realistic shadowing, we'll add the global illumination effect, ambient occlusion. To enable global illumination effects, (ambient occlusion in this case) we must create a RenderMan environment light from the *RenderSettings-> Features* tab. Enable *RayTracing* and create an *Environment Light*. The environment light is where we access all our setting for occlusion. Tune the settings as shown, hide your other lights, and render. You should get a classic occlusion result. (If not, make sure your *Max Dist* is set to "50".) Tune your settings until you are happy.

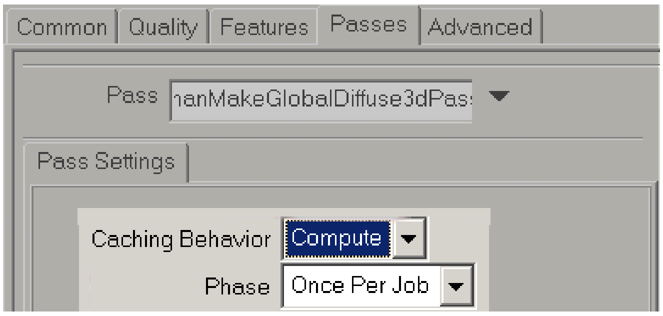


Bias

Gain

Bake rmanMakeGlobalDiffuse3dPass

Environment Light



Pass ʼnanMakeGlobalDiffuse3dPas:

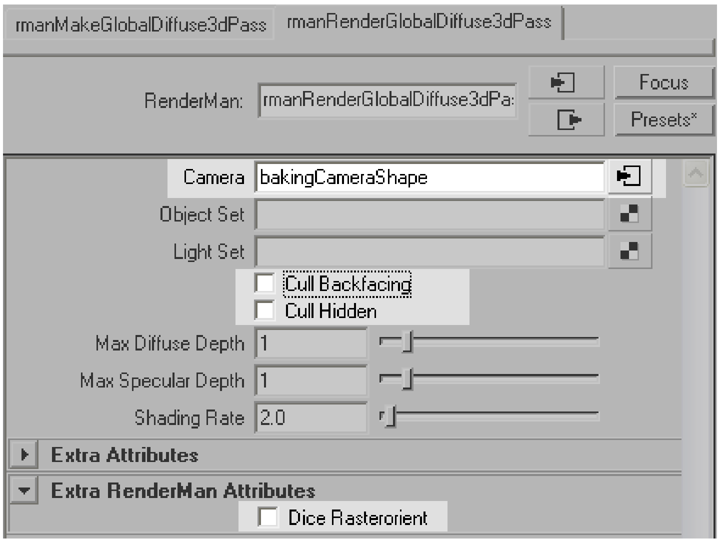
Pass Settings

Caching Behavior Compute

Phase Once Per Job

Bake Pass

18. To save recalculating this expensive ray tracing effect every frame, we are going to bake the calculation into a brickmap. First, create a new *MakeGlobalDiffuse3d* pass in the *Bake* field of the Environment Light. This will create a new pass in the *Render Settings -> Passes* tab where you can tune the *Caching Behavior* and *Phase* for the pass. We will use *compute* and *once per job,* as shown above.



RenderMan: rmanRenderGlobalDiffuse3dPa:

Camera bakingCameraShape

Object Set

Light Set

☐ Cull Backfacing

☐ Cull Hidden

Max Diffuse Depth 1

Max Specular Depth 1

Shading Rate 2.0

Extra Attributes

Extra RenderMan Attributes

☐ Dice Rasterorient

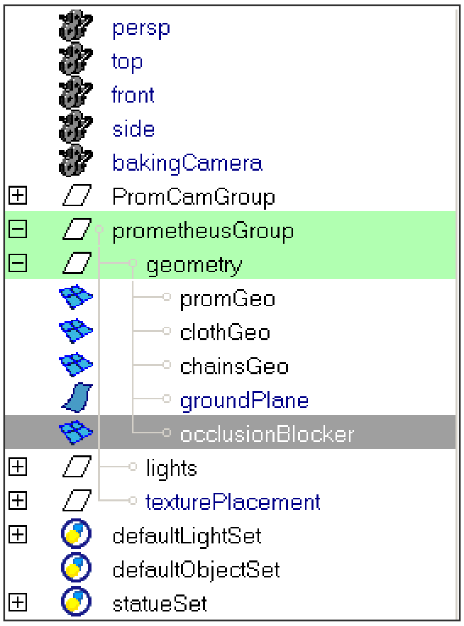
Configure the RenderDiffuse3d pass as shown

19. The other tab created with the new *MakeDiffuse3d* pass is the *RenderDiffuse3d* pass, found beside the former tab in the Render Settings. Similar to the SSS setup, this tab is where we setup a camera for

final baking. Specify the same *bakingCameraShape,* and then disable *Cull Backfacing* and *Cull Hidden*. Add the RenderMan attribute *Dice Rasterorient* and disable it. Render to bake, then set *Caching Behavior* in the *MakeGlobalDiffuse3d* tab from *Compute* to *Reuse*.

●Tip

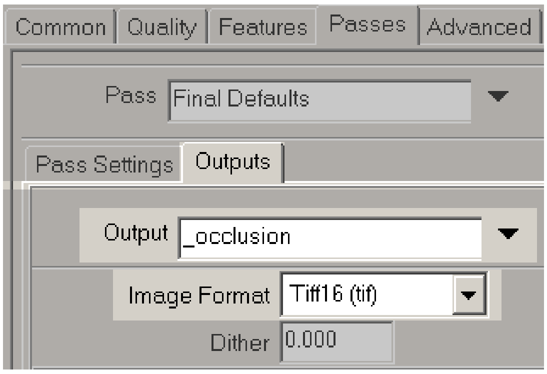
Quality vs. Speed: The environment light has a number of parameters for tuning the type, quality and speed of the global illumination effects. *MaxVariation* controls the amount of interpolation between pixels and is analogous to *Shading Rate*: try setting this to 1 (or less) for production quality and 10 (or more) for preview rendering.



20. Now we will simulate the illumination of the skylight. We will fake an area light using ambient occlusion, because ambient occlusion makes it easy to bake this data into a brickmap (unlike an actual area light). Hide your first environment light, and unhide the *occlusionBlocker* mesh in the geometry group. This missing face at the top will act as the skylight letting the occlusion pass see white. Next create new environment light (a second environment light) from the *Render Settings-> Features* tab by right-clicking on the *Environment Light* field.

21. We will make this Environment Light act like an area light by setting the *Max Distance* attribute to a value larger than the maximum dimension of our blocking geometry. Here a value of 10,000 is used. The other settings are the same as the last Environment Light.

22. Use similar bake settings for this pass. Remember to turn off *Cull Backfacing*, *Cull Hidden* and *Dice Rasterorient* and specify the *bakingCameraShape*. Now render the scene to bake the occlusion brickmap. You should see this brickmap file, along with the other brickmaps, ptc files, and shadow maps is your */renderman/$scene/data* folder in your current project (these default directories can be changed in the *Render Settings-> Advanced* tab).



Pass Final Defaults

Output _occlusion

Image Format Tiff16 (tif)

Dither 0.000

Add secondary outputs to the Final Pass

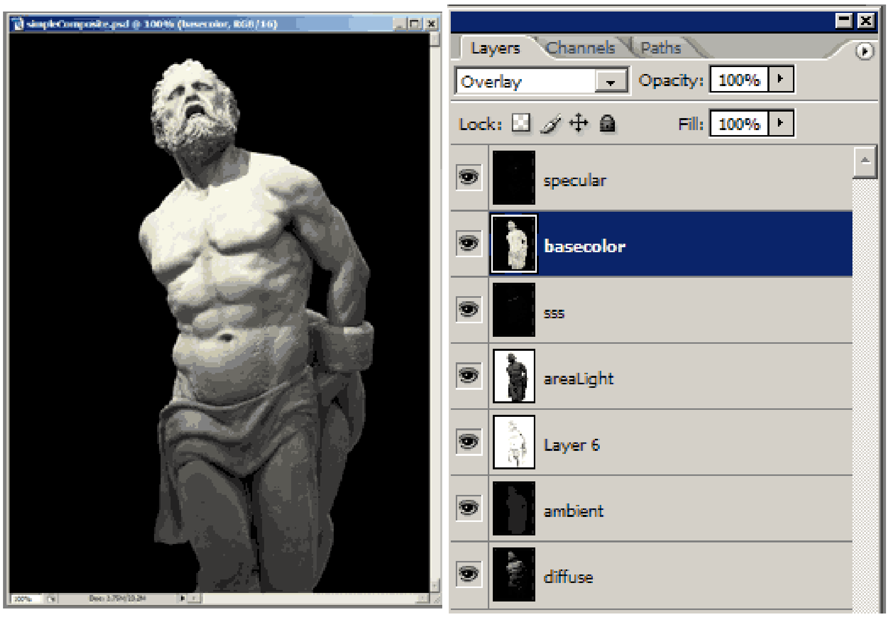
23. Now we will create the secondary outputs for compositing the final image. First unhide all the lights, except the fist environment light which should remain hidden. To prepare output passes we go to *Render Settings-> Passes* tab. Here we add the following secondary outputs to our *Final Defaults* pass: diffuse (*DiffuseDirect*)*,* occlusion *(OcclusionDirect), ambient (Ambientlight),* specular *(SpecularDirect),* subsurface scattering *(Subsurface), &* surface color *(DiffuseColor)*. Add these passes

by clicking on the black arrow next to *Pass* and selecting *Final Defaults*. Next click on the black arrow next to the *Output* field and select the outputs one at a time. Set all outputs to format *Tiff16*. Render once in batch mode and look at your secondary outputs, which will appear in the Maya project directory, *renderman-> prometheus-> images*. To get our original "close occlusion" pass, unhide our first envLight and hide the second "skylight" envLight. Render with only occlusion as a secondary output, and to do that you must delete the other secondary outputs by selecting and deleting them from the pull-down menu. To prevent the first occlusion pass from being overwritten, change its name to *occlusion_arealight.tif*.

●Tip

Secondary Outputs: Rendering your scene in separate passes give you maximum flexibility in compositing and greatly reduces the amount of time you need to spend tuning your renders in 3D. RenderMan efficiently generates all the passes you need to rebuild a final render in compositing. Passes are known as secondary outputs in RenderMan, and the renderer is very efficient about outputting these component parts from a single render (any number of outputs are created at one time). Once you start using secondary outputs in compositing and you won't go back. When you break your output down into its component parts: diffuse, reflectivity, occlusion, shadow, etc., you have greater flexibility in color correcting individual layers to more precisely match background plates in hue, saturation, expose, etc.

6 COMPOSITING



simpleComposite.psd @ 100% (basecolor, RGB/16)

Overlay

Opacity: 100%

Fill: 100%

specular

basecolor

sss

areaLight

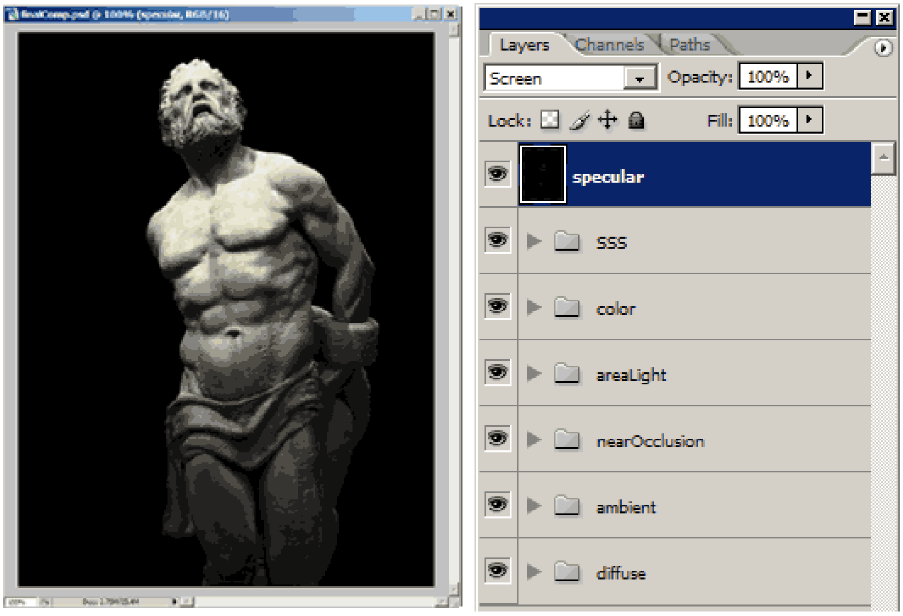
Layer 6

ambient

diffuse

Layer the secondary outputs with diffuse on bottom

24. The power of rendering with secondary outputs is that it gives you incredible control over tuning the look of your plate in 2D. For this example, we will use Photoshop to demonstrate the basic steps used to composite our secondary outputs, although in a studio compositing might be performed with a dedicated compositing program. For user's of RenderMan Studio, the image tool "it" may be used to perform compositing (see below). Now using Photoshop, open all your secondary outputs and layer them in one image as shown above.



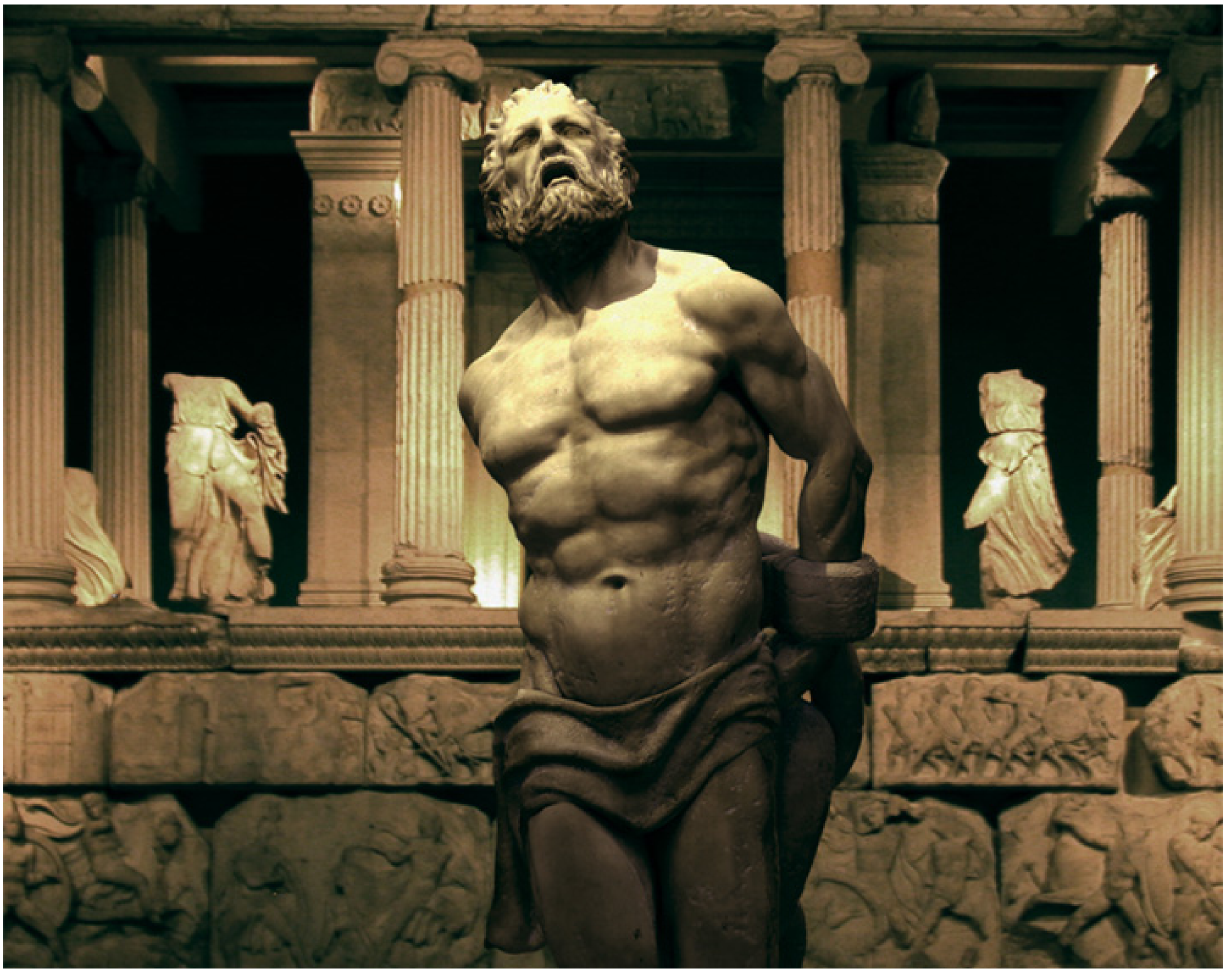
Screen

Opacity: 100%

Fill: 100%

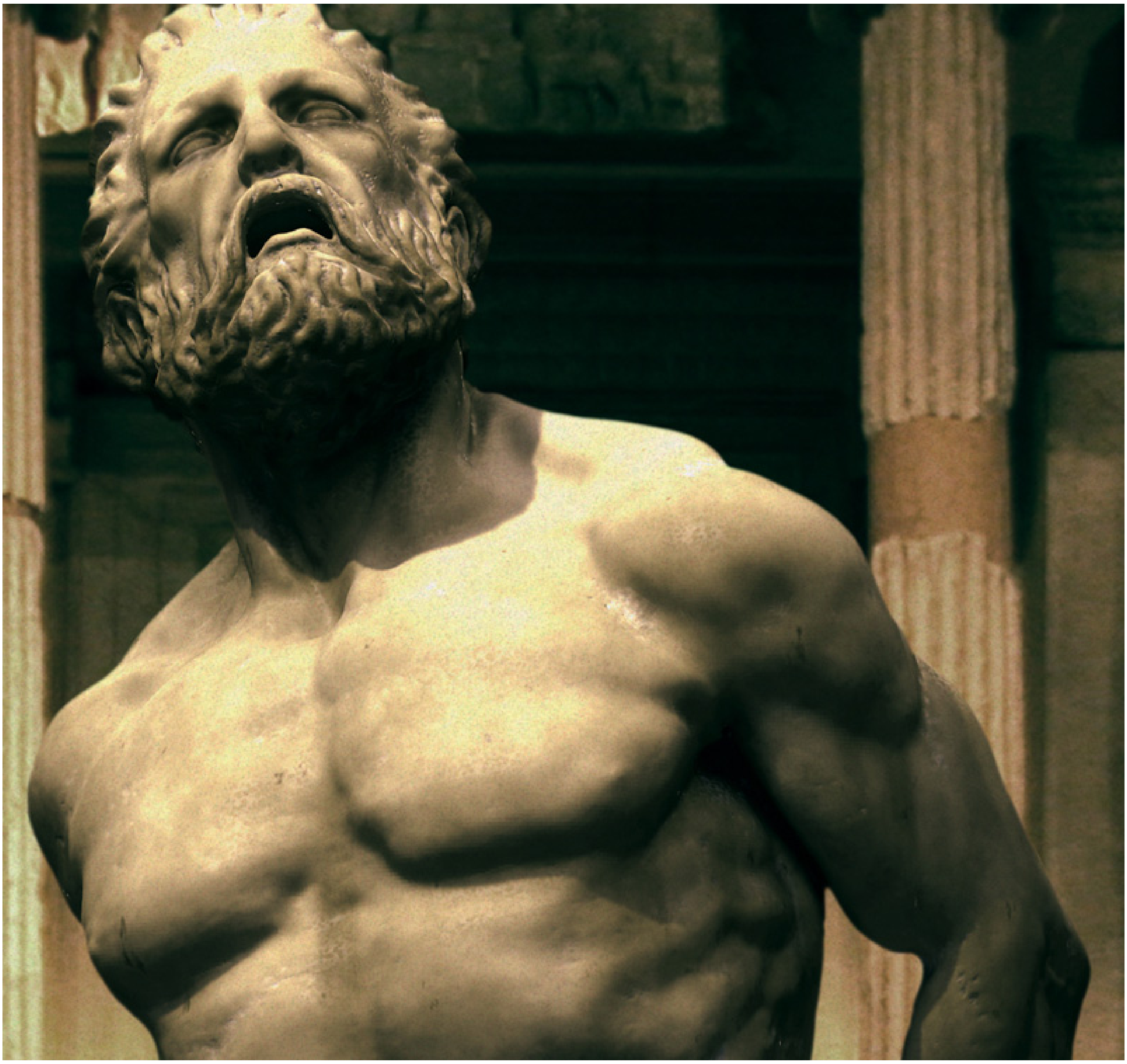
Adjusting lighting in compositing can be much faster than rerendering

25. Now we need to set the blending modes between layers. *Diffuse* is the lowest layer and above it screen on the *ambient* channel. Next multiply by the close-occlusion layer. Then either multiply by the area light occlusion to take light out of the image, or screen to add light. Then overlay the *surfacecolor* pass. Finally screen both the subsurface and *specular* passes. Use level adjustments to tune the image.



The final composite

26. Finally we color correct our plates to match the background. The main challenge of color correction is finding the correct transition from light to shadow and then matching the color temperature in the highs, mids, and shadows. Here the background plate is very warm red-orange in the mids and highs, while the shadows have more cyan. The final match was done using a combination of color balance for the temperature and levels for transitions.



Closeup detail

About the Author: **Scott Eaton** is currently Creative Technical Director at **Escape Studios** in London, where he divides his time between production and lecturing. His client list includes Sony, Microsoft Game Studios, The Mill, Double Negative, and many other leading production houses and games studios. Scott received his master's degree from the MIT Media Lab and subsequently studied traditional fine art at the Florence Academy of Art, Italy.

Prev | Next

Pixar Animation Studios

RenderMan Educational Videos

Overview

These educational videos provide good introductions to some different features and techniques available with RenderMan for Maya. They are provided by **Digital-Tutors**, from their *Introduction to RenderMan for Maya* CD.

GETTING STARTED

"Fast Start" Video A great introduction to RenderMan for Maya . . . covering everything from loading the plug-in, to basic operation, to the introduction of some advanced features. You can download the Maya scene for this tutorial **here**.

TIPS & TRICKS

RenderMan for Maya Mattes Make holes in things with RenderMan Attributes.
Rendering NURBS Surfaces When it comes to NURBS and Subds, RenderMan renders true curved surfaces, always smooth!
Sharing Geometric Attributes Learn how to share RenderMan attributes between objects.
Using RenderMan and Maya Cloth Render better looking cloth)with subdivision surfaces!

ADVANCED TECHNIQUES

Creating Depth of Field Effects You deserve physically accurate depth of field.
Rendering ZBrush Displacement Maps in RenderMan for Maya Make the most of RenderMan's powerful displacements!

RfM Fast Start

RfM Fast Start Video

This video is a great introduction to RenderMan for Maya . . . covering everything from loading the plug-in, to basic operation, to the introduction of some advanced features. You can download the Maya scene for this tutorial **here**.

This educational video has been provided by **Digital-Tutors**.

Tutorials

Documentation Files

You can follow along with tutorials by using the Maya files included with the documentation. The tutorial files are collected in a Maya Project and can be located in your documentation here **doc_location / Recipes_and_Tutorials / rfm_project**. Use the folder called ***rfm_project*** as a Maya Project directory, update Maya's project location, and you'll be ready to go.

Online Access to Tutorials

The tutorial files may also be accessed online in the **Software Download Area**. Simply download "Tutorial Files" and expand it. Use it as a Maya Project.