Features and Effects

Navigate using the menus on the left.

Rendering

Prev | Next

Rendering Passes

Introduction

Fundamentally, RenderMan renders images using passes. Just about any but the simplest of scenes will require multiple passes to render: a shadow pass for shadows (deep or not), a `txmake` pass for textures, and so on, all leading to the final pass for your fully rendered image. Most of these passes are pre-defined and happen behind the scenes without requiring you to do anything you set up your scene and RenderMan does the rest but you can also take advantage of passes by creating them yourself.

RenderMan for Maya provides sophisticated machinery for creating and managing passes. During a render job, pre-defined passes can be generated to compute data before the final image is rendered. This data can be referenced during the rendering of the final image, and it can be cached to use in subsequent renders, if you so desire. Pre-computed passes can be used to create everything from depth maps to reference images to occlusion passes.

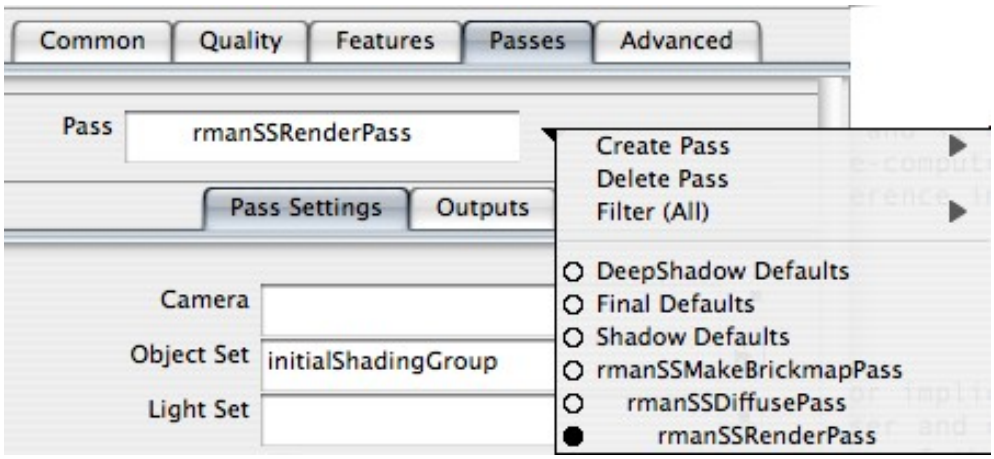
Creating Passes

Passes are created either explicitly or implicitly. Explicit passes are created purposefully by the user and can be used for computing all sorts of calculations before the rendering of the final frame. The generation of *reference image* passes may be one of the most common and useful pre-computed passes.

To create a pass explicitly, open the Passes Tab of the Render Globals. From the pull-down menu next to the *Passes* field, select the type of pass to be created:

```
Pass Menu: Create Pass-> Pass Type
```

Implicit passes, on the other hand, are created automatically by RenderMan for Maya as the need arises. For example, when creating a subsurface scattering effect, RenderMan for Maya automatically creates a sequence of passes to generate the effect: a pass for generating a point cloud, another for filtering the point cloud, and a pass to make the brickmap so it can be used in the beauty (Final) pass.



Managing Passes

All passes can be configured independently of other passes. Each pass has its own custom settings that override the settings in the Render Globals. Any individual pass can have its own quality settings, its own pass-specific camera, or many other kinds of independent behaviors as defined in the pass tab.

All passes (whether explicit and implicit) can be controlled via the Passes Tab of the Render Globals. The *Settings* sub-tab provides controls for the pass that is currently selected in the passes tab. Additional attributes may be added to passes to provide additional control.

Secondary Outputs

With RenderMan for Maya any pass in a render job can output its own *secondary outputs*, created during the primary pass. RenderMan for Maya has a flexible advanced system for constructing passes and secondary outputs. The most common use of secondary outputs is the creation of additional image elements for compositing, like specular, beauty, diffuse, etc. For more information about secondary outputs refer to the Secondary Outputs document.

Strategies

There are several strategies for computing pre-passes. Proper setup of pre-passes can have a substantial impact on render speeds. The most important pass settings are described below.

Caching Behavior
It is important to set the caching behavior for the shot. There are three types of caching behavior: *compute*, *reuse*, and *disable*.

Compute Causes the pass to be computed every time the frame is rendered, whether or not a pass has previously been rendered.

Reuse Causes previously cached data to be used again. No additional calculations are peformed in *reuse* mode, but the cached data must exist on disk.

Disable Simply disables the pass, causing the entire pass to be ignored.

●Tip
When tweaking shaders during iterative renderings, the caching behavior of shadow maps can often be set to *reuse*. By reusing shadow maps, the shadows only have to be calculated once and are reused for

each iterative rendering. No more waiting for shadow map calculations.

Phase
Most passes have a *Phase* parameter that determines how often a pass is calculated. The choices are: *Once Per Job* and *Every Frame*.

Once Per Job Calculates the pass for the first frame of an animated sequence.

Every Frame Calculates a pass for each frame of an animation.

For single frames, both settings have the same effect. When rendering animated sequences is where the *Once Per Job* setting can be advantageous.

How often a pass needs to be computed is dependent on the shot. For shots where the elements in a pass are not moving from frame to frame, *Once Per Job* would be the right setting. However, in cases where objects in passes are moving and deforming during a shot, each frame would require a new calculation. In those cases *Every Frame* would be the correct setting.

●Tip
It is common to use the *Once Per Job* setting with shadow maps. For instance if the elements in a shadow pass are not moving during a shot (like 3D models of skyscrapers) then only one shadow map needs to be computed for the entire shot. By using the *Once Per Job* setting (as opposed to the every frame setting), the scene will render much faster, more efficiently, and require significantly less disk resources.

Sets
Controlling the membership of objects rendered in pre-passes can provide additional optimizations. Unnecessary calculations can be avoided by selecting only those objects that are required to participate in the pre-computed pass.

To control this set behavior, create a Maya set that only includes the appropriate geometry and enter the name of the Maya set in the *Set* parameter of the pass.

●Tip
Images that aren't moving in a shot can be put into one Maya set and that pre-pass can be calculated *Once Per Job*. Moving objects can be stuck into their own Maya set.

2D Textures Vs. 3D Point Clouds

RenderMan for Maya is capable of caching pre-pass data as 2D textures or as 3D point cloud files. Whether a pre-computed pass is a 2D texture or a 3D point cloud is largely transparent, but it does have some impact on workflow.

2D Texture Passes

Most pre-computed passes are cached into 2D textures. Some 2D textures, like shadow maps, also contain depth values, but are fundamentally 2D textures.

3D Point Clouds

Pixar has developed the technology of "3D textures", which are basically point clouds (color data stored in x, y, and z). RenderMan for Maya automatically bakes certain types of computations into 3D point clouds. Subsurface scattering is an example of this. Other calculations, like occlusion, can be baked into either format. By default, occlusion is baked into 3D point clouds, but it can be baked into 2D camera projected textures instead. For instance, 3D point clouds can occasionally consume much more disk space that 2D textures would, and in that case 2D textures would be preferable.

RenderMan for Maya will generally choose the proper format for baking data, but RenderMan for Maya also provides controls for additional customization.

Conclusion

RenderMan for Maya provides advanced controls for independently configuring individual pre-passes of a render job. Proper management of pre-passes can dramatically optimize and accelerate rendering.

Prev | Next

Secondary Outputs (AOVs)

Introducing Secondary Outputs

Another feature of RenderMan for Maya is the ability to generate secondary outputs (also called Arbitrary Output Variables, or AOVs). While rendering, RenderMan for Maya can output additional data (specular highlights, reflections, shadows, etc.) into separate image files, called secondary outputs. These outputs are not rendered in multiple passes, but are created at the same time the main image is being rendered

Secondary outputs are useful for compositing. It can be more efficient to adjust certain effects in compositing (such as tweaking specularity) than rendering the image over and over again. Secondary outputs can also be used to create special effects. RenderMan for Maya provides controls to generate exactly the kinds of secondary outputs required.

For more information about workflow associated with creating secondary outputs refer to the **Secondary Outputs Tutorial**. What follows here is an overview of the basic outputs and how they are combined to create your final image.

About Outputs

Here is a list of shadingmodel outputs. These additive terms are required to reconstruct the beauty render. They all must be passed through layering operations.

`AmbientConstant`

: The (surface?) shaders' constant "built-in" ambient illumination

`AmbientLight`

: Illumination from ambient lights

`Backscattering`

: Total contribution of backscattering effects

`DiffuseDirect`

: Unoccluded diffuse response to direct lights

`DiffuseDirectShadow`

: Diffuse response that is shadowed (subtract this from DiffuseDirect)

`DiffuseEnvironment`

: Diffuse response from environments

`DiffuseIndirect`

: Diffuse response reflected from other objects

`Incandescence`

: Total contribution of incandescent effects

`Refraction`

: Total contribution of refraction effects

`Rim`

: Total contribution of rim effect

`SpecularDirect`

Specular response from direct lights

SpecularDirectShadow

Specular response that is shadowed, subtract this from SpecularDirect

SpecularEnvironment

(Unoccluded) specular response from environments

SpecularIndirect

Specular light reflected from other objects (coherent reflections)

Subsurface

Total contribution of subsurface scattering

Translucence

Total contribution of cheap/thin translucence

The recipe for compositing your beauty image is a simple additive process using these outputs:

```
result = SpecularDirect - SpecularDirectShadow + SpecularIndirect +
         SpecularEnvironment + AmbientConstant + AmbientLight +
         DiffuseDirect + Translucence - DiffuseDirectShadow +
         DiffuseIndirect + DiffuseEnvironment + Backscattering +
         Subsurface + Rim + Refraction + Incandescence
```

Additionally, there are ancillary outputs that are not used to create the beauty render but can be useful for other compositing tasks:

SpecularColor

Color used to scale specular illumination

DiffuseColor

Color used to scale ambient and diffuse illumination

OcclusionDirect

Occlusion of direct illumination (shadows)

OcclusionIndirect

Occlusion of indirect (ambient) illumination

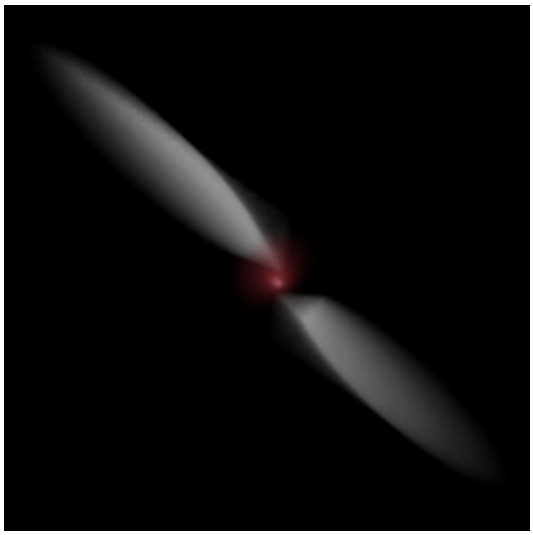
Special RenderMan Camera Effects

Introduction

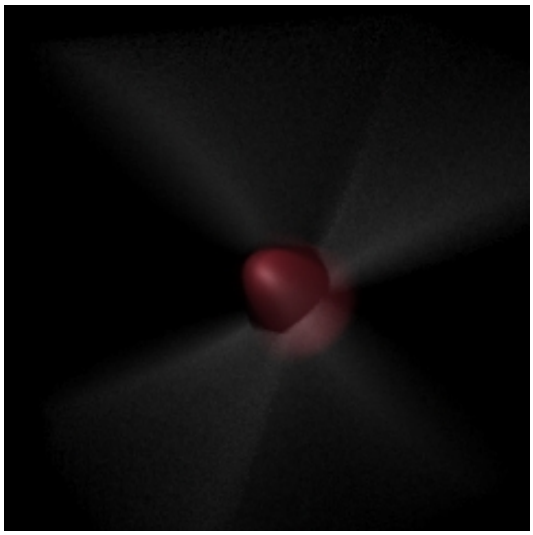
RenderMan provides users with big time camera effects for your use and enjoyment, including highest-quality motion blur, depth of field, and special effects implemented to help you simulate imperfections in your camera lens to enhance your efforts to create photorealistic renders.

Motion Blur

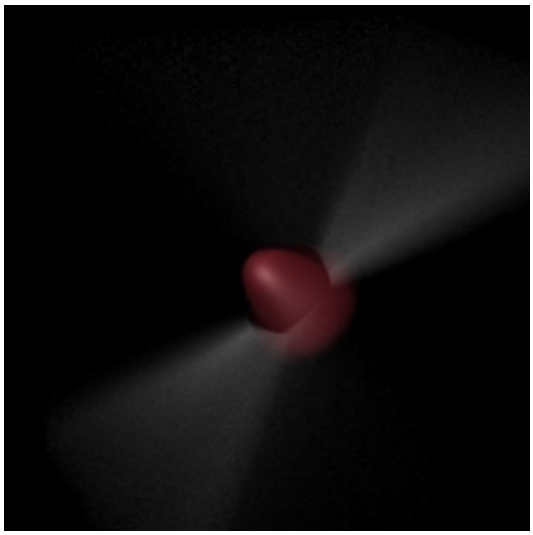
Pixar's RenderMan can motion blur until the cows come home, offering users unparalleled quality and control. The RenderMan for Maya plugin offers simple control over motion blur effects via the *Render Settings*, including the new shutter open and close settings introduced in version 13 of the renderer. Additionally, multi-segment motion blur can be enabled by adding the *Motion Samples* attribute to the blurred object's Transform node.

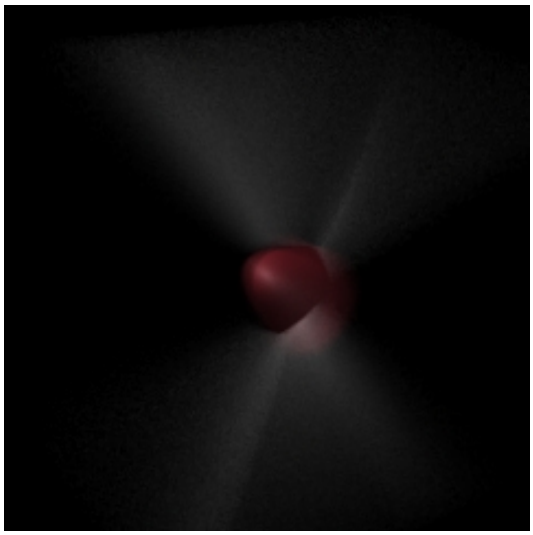


Motion Blur!



Multi-Segment Motion Blur!



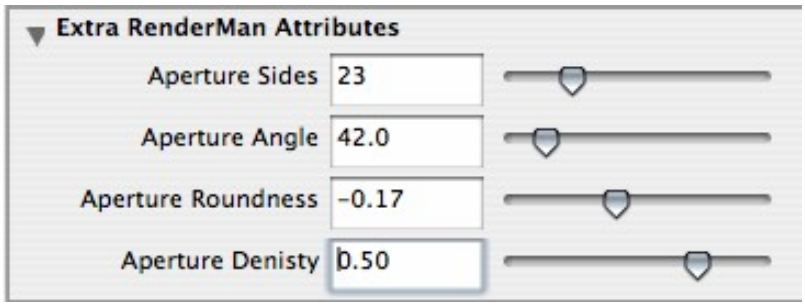


Multi-Segment Motion Blur
Slow Shutter Open, Fast Close

Multi-Segment Motion Blur
Fast Shutter Open, Slow Close

Depth of Field and Bokeh

The RenderMan for Maya plugin uses the Depth of Field settings for your Maya camera. In addition, you can add *Aperture Controls* to your camera via the Attributes menu in the Maya Attribute Editor. The Aperture Controls allow you to manipulate the shape and other characteristics of your lens , simulating the imperfections found in an actual camera's lens.



RenderMan Aperture Control Attributes

Prev | Next

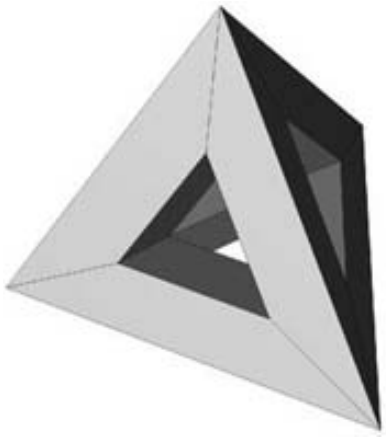
Geometry

Geometric Primitives

Subdivision Surfaces

RenderMan for Maya supports Maya subdivision surfaces (available in Maya Unlimited), including support for hierarchical editing, which allows for exact control over components of the subdivision mesh at finer levels of subdivision. Also, since RenderMan renders true curved surfaces, tessellation is never an issue. RenderMan handles subdivision surfaces well.

The subdivision surface is a relatively new geometric primitive which combines strengths of both NURBS and polygons. These combined strengths make subdivision surfaces well suited for modeling complex objects, especially organic objects that require animation, like characters.



A polygon model.

A subdivision surface is completely smooth.

A subdivision surface is described by a control mesh of points, like a NURBS surface. Additionally, a subdivision surface's control mesh is not confined to being rectangular, which is a major limitation of NURBS. In this respect, a subdivision surface's control mesh is analogous to a polygon model. But where polygon models require many facets to approximate being smooth, a subdivision surface *is* smooth; it is a true curved surface, meaning that Pixar's RenderMan actually renders the high-level subdivision surface (and only tessellates on a pixel basis). Subdivision surfaces will never have a faceted look (like polygon models can), no matter how the surface animates or how closely it is viewed.

A subdivision surface can be constructed to make highly efficient use of geometry, putting geometry only where it is needed and not where it's not a great benefit for animation.

Subdivision Surface Strengths:

1. A true curved surface (like NURBS) with unique support for creases and points.

2. Animation No patch cracks (like can happen with NURBS). Put geometry only where needed (unlike NURBS and polygons).
3. Displacements High quality. (No cracks, a common artifact with displacements on polygons.)

With RenderMan, rendering a complex subdivision surface is much more efficient than rendering the equivalent polygon model (a model which would require many model facets to approximate a smooth surface).

(top)

NURBS

RenderMan for Maya provides complete support for Maya NURBS, including trimmed surfaces. NURBS (or nonuniform rational B-splines) are essentially rectangular 2D patches which are stretched and bent into 3D shapes. Because RenderMan renders true curved surfaces, tessellation is never an issue; NURBS are always smooth.

Note

Maya uses U & V coordinates to parameterize NURBS surfaces, while RenderMan similarly uses S & T. RenderMan for Maya correctly interprets S & T values automatically. This would only be an issue when importing custom RenderMan shaders.

(top)

Polygons

RenderMan for Maya provides full support for Maya polygons, including Maya's UV mapping tools. A single polygon is a face, with any number of edges and corners (vertices). A polygon mesh is composed of many individual polygons, a polygonal mesh. Polygons tend to be easier to model than NURBS since a polygonal mesh can be any arbitrary topology, whereas NURBS must be carefully organized patches. Polygons have several inherent disadvantages, however, when compared to NURBS or subdivision surfaces:

1) Polygons are susceptible to faceting artifacts

2) Polygons require dense geometry to approximate smooth surfaces

3) Polygons do not displace satisfactorily (due to their essentially discontinuous topology)

Note

For users of Maya Unlimited, there are often benefits to be gained by converting polygons to subdivision surfaces, especially when displacements are being used. The caveat here is to make sure that the polygonal geometry is not too dense, which

could create an inefficient subdivision surface.

(top)

Prev | Next

Instanced Geometry

Support for Instancing

RenderMan for Maya supports Maya's instancing of geometry. Instanced geometry is a reference to an original geometric master and is particularly efficient, being only a reference to the original object. Using instanced geometry can accelerate Maya's interactive display. Unfortunately, it also has some restrictions, inasmuch as any changes made to the original object affect the instance as well.

There are several added benefits to using instanced geometry with RenderMan for Maya. Instanced geometry is much more efficiently represented internally in the renderer. A scene rendered using Pixar's RenderMan with fifty folding chairs would be much less efficient than the same scene rendered with one folding chair and forty-nine instances. Instancing geometry can be a very useful technique for optimizing rendering.

Particles

Particle Support

RenderMan for Maya provides automatic support for Maya particles, including Maya particle instancing. RenderMan for Maya supports the Maya particle types listed **here**.

Software Rendering

RenderMan for Maya renders particles using pure software rendering. RenderMan's highly optimized particle primitives are fast and efficient. There are added benefits to software rendering over hardware rendering:

Benefits of software rendered particles:

1) Antialiasing, motion blur, and full integration into the scene

2) Particles can cast shadows

3) Arbitrary shaders can be attached

Also, when rendering particles through RenderMan you have control over the physical size of the particles in world space units. This is due to the fact that Pixar's RenderMan is resolution-independent, while hardware rendering modes commonly work in pixel units.

Per-Particle Effects & Arbitrary Attributes

RenderMan for Maya automatically establishes the correct rendering context for variations in particle color, opacity, and radius. You can write MEL expressions to calculate per-particle radii (**radiusPP**), color (**rgbPP**), opacity (**opacityPP**), and lifespan (**lifespanPP** and **useLifespanPP**). You can also use Maya to setup procedural ramps which remap these values.

Finally, you can create arbitrary attributes and associate them with the RenderMan representation of your particle objects. This might be of use if you plan to write a custom shader to perform calculations using per-particle or per-particle-object attributes. For more on the general mechanism for transmitting arbitrary primitive attributes through RenderMan, please refer to the primitive variables reference.

Particle Instancing

RenderMan for Maya supports Maya particle instancing. For complete details on how to set up your scene for particle instancing, please refer to the Maya documentation.

There are some cases where RenderMan for Maya does not correctly translate Maya's configuration. Generally, if you stick to the proscribed steps and avoid obscure orientation configurations, you'll get good results.

Fur and Hair

Introduction

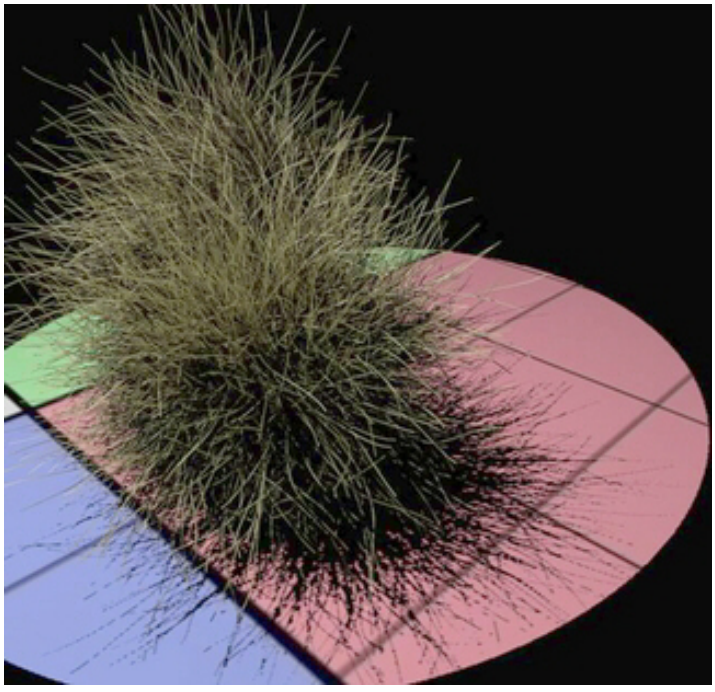
RenderMan for Maya supports Maya Fur and Hair (available with Maya Unlimited), for the creation of highly convincing fur effects. Fur and hair is fully supported, from keyframe animation to dynamic simulations. RenderMan for Maya will automatically render Maya fur and hair, requiring no additional setup.

RenderMan renders these effects fast and efficiently, although rendering fur can be an computationally expensive operation. In addition, both fur and hair are rendered as another scene element in software (RenderMan does not treat fur as a post-process effect), which allows full integration for anti-aliasing, shadows, etc.

Fur and Shadows

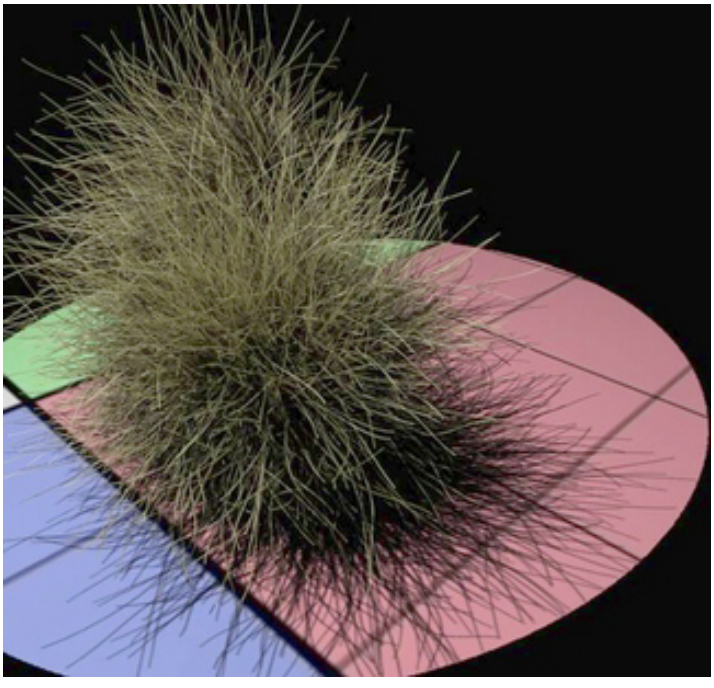
Shadows can give fur the added depth needed for exceptional realism, and Pixar's Deep Shadows are ideal for this purpose. Deep Shadows are capable of creating soft, subtle shadows, free of artifacts. They are more expensive than traditional shadow maps, but are well worth it when rendering fur, due to issues associated with using traditional shadow maps and fur. As shown in the image below, fur with traditional shadow maps has several undesirable artifacts, especially in animation. Deep Shadows, however, are perfect for use with fur.

Fur with Depth Map Shadows



In this image, the shadow on the floor has the artifacts that typically occur when using traditional shadow maps and fur. The only way to compensate for this is by dramatically increasing the resolution of the shadow map, which has its own issues. These artifacts will pop and flicker when animated.

Fur with Deep Shadows



With Deep Shadows, the shadows of the fur are much better looking. Deep Shadows allow the shadow to be filtered, eliminating blocky artifacts. Deep Shadows also can be semi-transparent, which adds additional character to the fur, and they look totally excellent when animated.

For information about fur workflow see the **Fur and Hair Tutorial**.

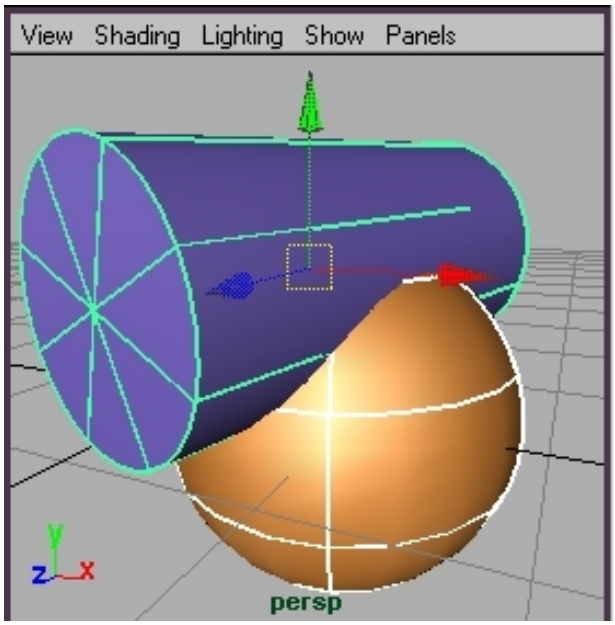
CSG (Constructive Solid Geometry)

RenderMan for Maya provides functionality for CSG Boolean operations on geometric surfaces computed in the renderer. CSG stands for Constructive Solid Geometry, allowing two, or more, objects (or groups of objects) to be combined in a number of ways, creating complicated geometry and special effects.

Using CSG requires that objects have RenderMan attributes attached to designate them as special CSG primitives. Then they must be grouped together and that group must have a CSG attribute applied to it as well. Here's how:

1 - Create Geometric CSG primitives

First create two pieces of geometry in Maya, as seen here. (For best results the geometry should be a closed **solid**, with no open holes.)



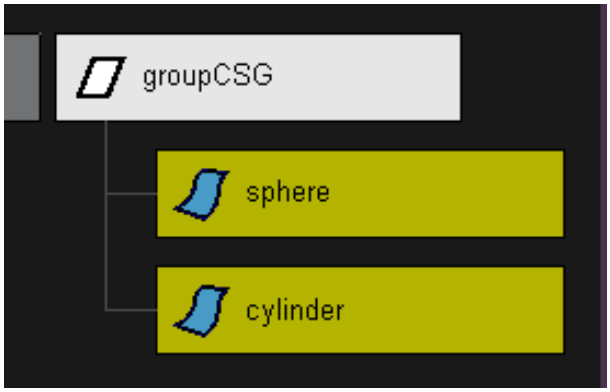
Next designate each geometric piece as a CSG primitve. Select the geometry (not the Shape node) and then from the Attributes menu, in the Attribute Editor:

Attributes-> RenderMan-> Add CSG Solid Type

This will add a dropdown menu to the Extra RenderMan Attributes. The objects at the base of a CSG hierarchy should always be designated as a CSG *Primitive*.

2 - Group and Apply CSG Operation

Group the two objects together. Now apply a CSG operation on the grouped node, in the same way you did to each piece of geometry.

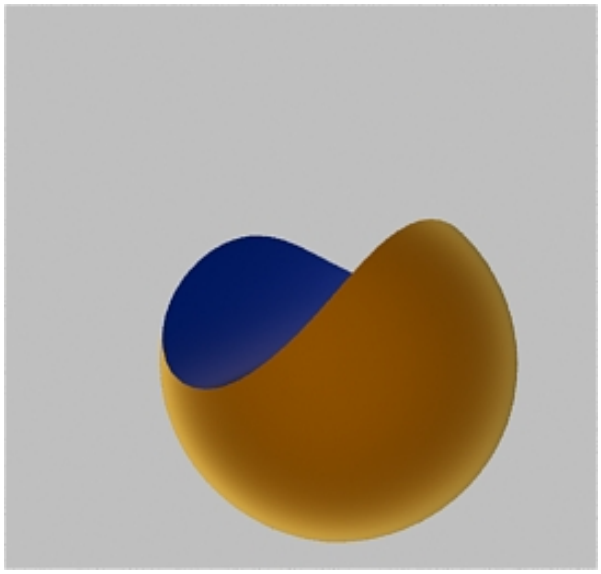


3 - Render "Difference"

Select *Difference* as the Solid Type for your grouped node. Render the scene.

The final image give us a CSG difference, with the cylinder making a hole through the sphere. In this case the hole is made in the sphere since it is the first item in the group. If the cylinder was first, then the sphere would make a hole in the cylinder. An objects order in the hierarchy of the group determines how objects are subtracted.

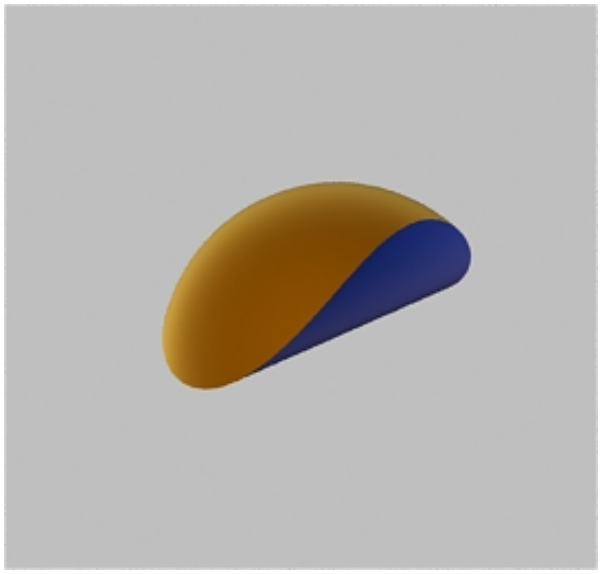
Note: The section the cylinder cuts away is shaded by the cylinder's shader.



4 - The "Intersection"

Change the Solid Type to *Intersection*. Now render.

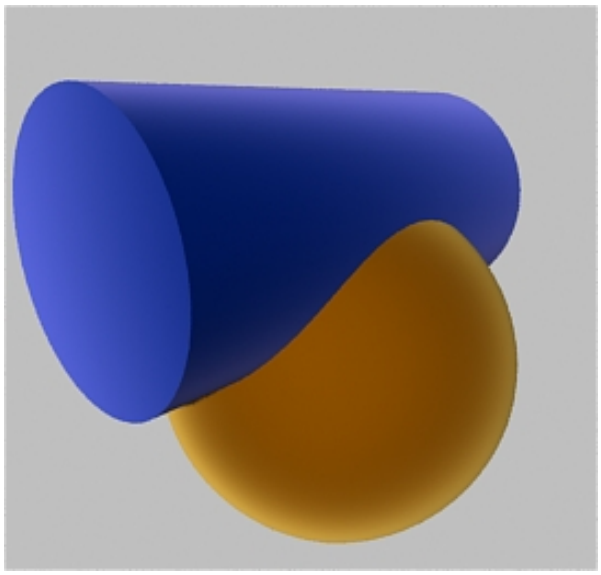
An object is formed from only the area that the objects both occupy.



5 - The "Union"

Finally, use the third operator by changing the type to *Union* Render. A single object is created from the intersecting objects.

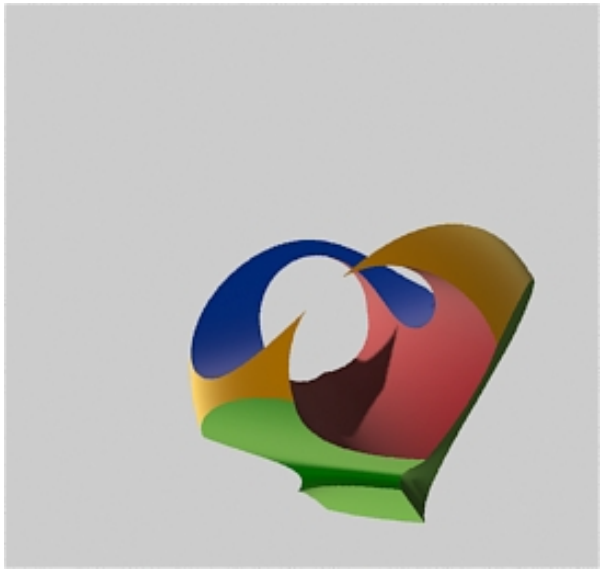
Note: These two pieces of geometry are now "fused" together into one piece. This can be useful in cases where the fused geometry will be used to perform other CSG operations on other geometry.



6 - Conclusion

CSG is a powerful technique for the following reasons:

- Groups of objects, entire characters or vehicles, can be added to CSG operations. (Be sure to designate all components of a object hierarchy (such as a character) as CSG primitives, grouping them with "Union" operators.)
- Easily animated, including motion blur.
- Multiple primitives can be grouped in a CSG operation.
- Hierarchies of consecutive CSG operations can be constructed. (For example, an "intersection" can be applied to a group of primitves and that result can be used in a "difference" operation against another primitive, and even that can be subtracted from another object.)



The image to the right was created by adding the cylinder and sphere to a larger hierarchy of CSG operations. We'll leave it as an exercise to the reader to figure out how this bizarre geometric artifact was accomplished.

Prev | Next

Understanding Shaders

Shaders and RenderMan

Introduction

At the core of RenderMan is the versatile RenderMan Shading Language. What follows is a discussion of some related key concepts which may benefit users of RenderMan for Maya. Because Maya Materials are automatically converted this knowledge isn't absolutely required, but it is valuable for understanding how shaders are implemented in RenderMan and will be of benefit for those who wish to create well-executed shaders.

All shaders answer the question, "What's going on at this point?" That is, a shader determines the color of a point by running algorithms to gather information about certain scene elements. The RenderMan Shading Language allows shaders to ask very complicated questions about what is happening at any given point. With the RenderMan Shading Language, shaders can be created without limitations, familiar shading models can be extended, or totally unique shaders can be developed. Physical properties of materials can be simulated or flat out defied in order to deliver whatever look is desired. The effects of studio lighting can be constructed by building lights that have special lenses, concentrators, flaps, or diffusers. Material types can also be combined, simulating the many coats of paint or finish applied to a surface. Remarkably realistic images can be produced with a few fairly simple shapes which have shaders that are asking the right questions.

Shaders can be generated in a number of ways. RenderMan for Maya renders Maya Materials and shaders created by Slim. Alternatively, shaders can be written using the RenderMan Shading Language for custom purposes, which can then be imported into RenderMan for Maya. Either way, high-quality shaders can be created, no matter which workflow is preferred.

Note

Custom RenderMan shaders can be imported into RenderMan for Maya. Once imported into Maya the parameters of these custom RenderMan shaders can be animated, but other Maya Materials cannot be wired into them. These RenderMan shaders can be connected to a top-level shading group, but these connections are constrained by the three available slots: surface, displacement, and volume.

Shader Types

RenderMan understands five types of shaders:

Surface Shader: Surface shaders are attached to all geometric primitives and are used to model the optical properties of materials from which the primitive was constructed. A surface shader computes the light reflected in a particular direction by summing over the incoming light and considering the properties of the surface.

Displacement Shader: Displacement shaders change the position and/or normals of points on the surface and can be used to place bumps on surfaces.

Light Shader: Lights may exist alone or be attached to geometric primitives. A light source shader calculates the color of the light emitted from a point on the light source towards a point on the surface being illuminated. A light will typically have a color or spectrum, an intensity, a directional dependency, and a fall-off with distance.

Volume Shader: Volume shaders modulate the color of a light ray as it travels through a volume. Volumes are defined as the insides of solid objects. The atmosphere is the initial volume defined before any objects are created.

Imager Shader: Imager shaders are used to program pixel operations that are done before the image is quantized and output.

Shader Space

RenderMan for Maya automatically translates Maya Materials, including associated projections. You might find that, in certain cases, these RenderMan shader space attributes are helpful (e.g. when using imported custom RenderMan Shaders or when an esoteric shader space like "NDC" is required).

Shader space is used to define how 3D procedural shaders and 2D projections are applied to objects in a scene. This is in contrast to using the natural parameterization of a surface (UV mapping) to map 2D textures onto objects. A 3D procedural shader emanates in three dimensions. Such shaders are often called *solid* shaders. Naturally, a point in 3D space must be given to a solid shader as the center starting point from which the shader expands. To define this point we use coordinate systems.

Now any node in Maya (object, light, etc.) can be used to declare a coordinate system for a procedural shader, but it turns out that the RISpec already contains a number of quite helpful predeclared shader spaces:

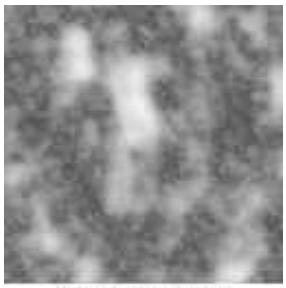
Coordinate System	Description

"object"	The coordinate system in which the current geometric primitive is defined. The modeling transformation converts from object coordinates to world coordinates.
"world"	The standard reference coordinate system. The camera transformation converts from world coordinates to camera coordinates.
"camera"	A coordinate system with the vantage point at the origin and the direction of view along the positive z-axis. The projection and screen transformation convert from camera coordinates to screen coordinates.
"screen"	The 2-D normalized coordinate system corresponding to the image plane. The raster transformation converts to raster coordinates.
"raster"	The raster or pixel coordinate system. An area of 1 in this coordinate system corresponds to the area of a single pixel. This coordinate system is either inherited from the display or set by selecting the resolution of the image desired.
"NDC"	Normalized device coordinates like "raster" space, but normalized so that *x* and *y* both run from 0 to 1 across the whole (un-cropped) image, with (0,0) being at the upper left of the image, and (1,1) being at the lower right (regardless of the actual aspect ratio).

Proceduralism

Introduction

One of the benefits of having a shading language is being able to generate patterns and textures from functions, such as solid fractal noise, ramps, or other useful functions (also known as procedural textures). Procedurals have several strengths over the the alternative for texturing objects (image maps). Ultimately, however, the decision on whether to use a procedural texture or an image map largely depends on circumstances. Sometimes procedural textures are the clear solution, sometimes images maps are more appropriate, and often a combination of the two is required.



fractal noise

Strengths of procedurals:

1) Procedurals can require less time to create. Image maps must be painted or photographed. Painting requires the time and talent of an artist. Photographs also require additional processing to be suitable textures. Procedural textures, on the other hand, are already to go (given a collection of pre-made fractal, ramp, and other stock procedurals). An original texture can be created quickly just by tweaking the parameters of a procedural texture.

2) Procedural textures have no pixel artifacts, like images are susceptible to. Zoom in too close to a texture map and you can see the original pixels. Procedural textures always look uniform whether viewed up close or far away.

3) Procedural textures spread out to infinity, with no repetition. Images do not. In order to cover large areas textures must repeat at some point and begin tiling. Tiling can produce artifacts when the same texture is repeated over and over. An acceptable repeating texture must be created by a trained artist. Procedural textures suffer none of these issues.

4) Procedural functions are simple to animate, just by key framing a parameter. Animating textures maps is much more difficult.

5) Procedural textures require little disk space, unlike high-resolution image maps.

Strengths of Image Maps:

1) Maps are easier to control. For instance, a specular map for a character's face must be shiny in certain places, the nose and cheeks, and less shiny in other areas. It's obvious that such a map must be painted, rather than constructed from procedural textures.

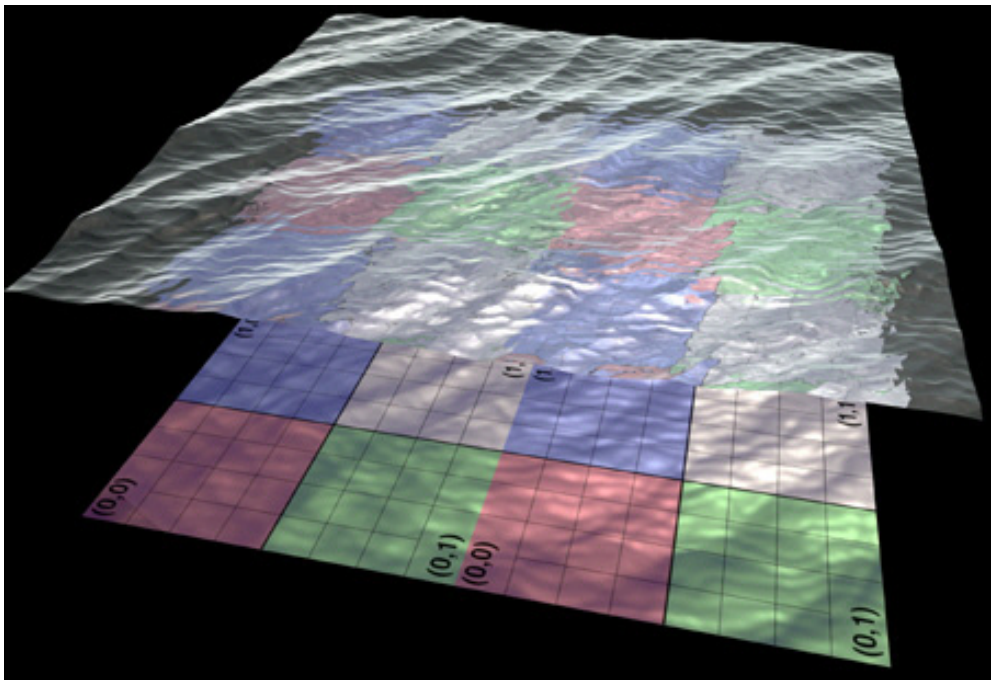
2) Maps are easier to edit. For instance, if the director demands that a dent or wood

knot must appear in an exact location, it is simpler to paint the detail directly . . . and an image is much easier to edit when the director decides to change the location later. It can be difficult to edit individual elements of a procedural texture without changing the entire pattern.

3) Fewer artifacts. When animated procedural textures can occasionally exhibit artifacts (popping, flickering, etc) caused by improper anti-aliasing. Image maps are already aliased. No problems there.

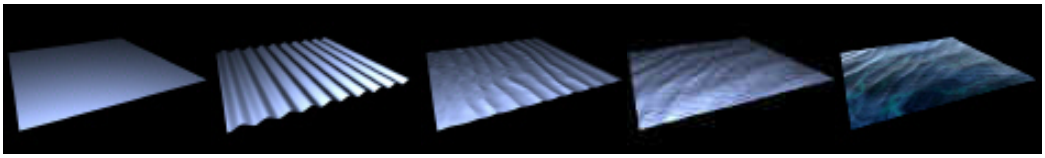
A Procedural Example

One of the strengths of procedural shaders is their ability to give simple geometry a realistic look and feel. Below, a flat plane has a complex animated procedural water shader attached to it. This final shader not only creates a physically accurate ray traced refraction, but it also displaces the geometry of the plane, effectively adding real geometric detail without modeling.



The water is an animated procedural shader

The water is built from layers of simple procedural textures, as can be seen in the smaller images. The water was created with an interactive editor and required no programming.



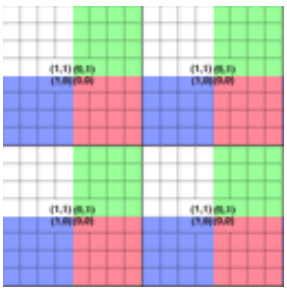
Simple layers are combined to create a complex procedural shader

---

Textures

Introduction

The following describes the use of textures in RenderMan for Maya. Texture files are often used in shaders, as an alternative to procedural textures.



Supported Formats

RenderMan for Maya accepts source textures in many image formats, including: TIFF, Alias, mayaiff, Radiance, JPEG, HDRI, and SGI RGB. RenderMan For Maya will maintain the precision of 32 bit floating point images by default.

Note

Source images for texture maps can be any resolution; however, RenderMan's conversion process causes these files to be resized into various other resolutions for fast filtered access, each being some even power of two resolution in both width and height. To avoid resizing (and the consequential interpolation of your pixels) use resolutions with a power of two: 256 x 256, 512 x 512, 1024 x 1024, etc.

Textures

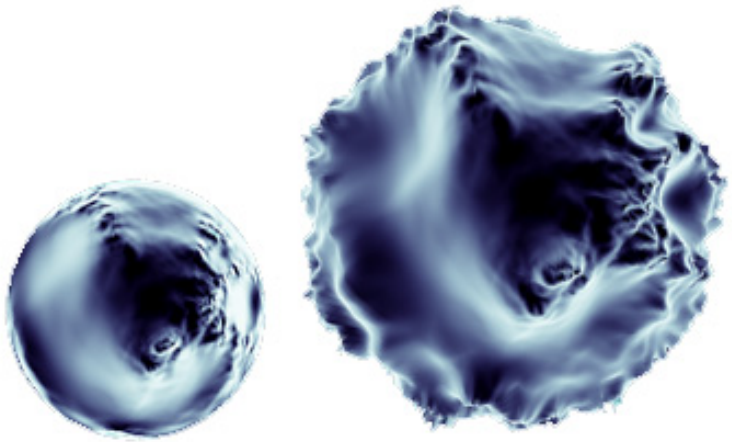
RenderMan handles textures very efficiently, and many textures may be used in a single scene. There is, in principle, no limit to the number of texture maps per surface.

Bump and Displacement Shaders

Introduction

RenderMan For Maya supports both bump and displacement shaders.

Bump



Bump *Displacement*

Bump mapping simulates roughness by perturbing the surface normal at any given point on an object. An object with a bump map appears to have pits and gouges, but the actually underlying geometry is unchanged. The normals are bent in ways that make the surface *appear* to be bumpy. The silhouette of a bumped surface will be the same as the original surface, as shown in the image on the right. Bump mapping can be useful, but when it is desirable to perturb the actual geometry of a surface, the more expensive option, displacement, can be used.

Displacement

One of the most interesting, powerful, and unique features of RenderMan is the ability to *displace* the surface geometry during shading. This allows the modeling of true dents, wrinkles, and other rough features which are only approximated by other techniques, such as bump maps. Since the geometry is actually moved by the displacement shader, not only the shading is changed, as in bump mapping, but the silhouette is changed and parts the surface can occlude other parts, just as they do in "real life." Displaced surfaces can also cast displaced shadows. Displacements can also be animated with interesting results. None of these effects can be achieved with bump mapping.

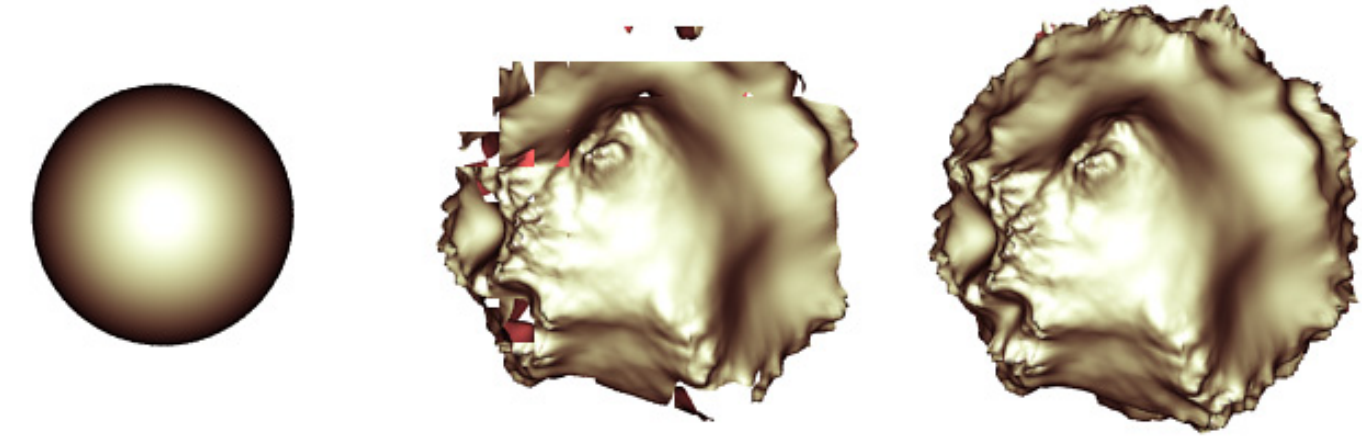
Displacements are sometimes tricky to use, however, and often lead to unexpected artifacts or performance problems. In particular, the fact that displacements are made to the surface geometry during the shading means that all of the code earlier in the rendering pipeline can only guess what the displacement might eventually be like. By following the guidelines presented here, it should be possible to eliminate most of the artifacts.

Avoiding Artifacts

Displacement shaders are evaluated and applied to the surface of objects during the shading stage. In fact, the displacement shader is the first shader that is evaluated on a surface, so that the subsequent light, surface, and atmosphere shaders are operating on the correctly moved data. The problem is, of course, that the various geometric operations which operate on the original surface geometry before shading (for example, bounding boxes and size estimation) don't know how much (if at all) the surface will move under the displacement. Therefore, the mechanisms that exist to eliminate geometric artifacts like polygonal silhouettes are occasionally outsmarted by the displacement.

Here are some of the artifacts that can be seen, and some hints on how to avoid them.

Displacement Bounds
All objects have a bounding box associated with them. This bounding box determines when the object is loaded into memory by the renderer, and it is usually tightly bound to the object in question. However, displaced geometry can be pushed outside of that box, and RenderMan doesn't pay any attention to an object until it reaches a pixel inside an object's bounding box, so things that are displaced out of the box will get missed, leaving large holes in the object. The following images show the artifacts associated with poorly set displacement bounds:



No Bump or Displacment *Incorrect Displacement Bound* *Correct Displacement Bound*

The displacement bound should be bound to the furthest extent of possible displacement (e.g. If your displacement has an amplitude of "1," the displacement bound should similarly be set to "1"). If you make the displacement bound too small, the object will still fall outside the bounding box and pieces may still disappear. If you make the displacement bound too large, then the bounding box will be too big and RenderMan will do far too much work processing the primitive and use up lots of memory storing the pieces that were done too early. Now we can see why getting the getting reasonable amplitude control are so helpful: because with both of those in place, figuring out the correct bound is much easier!

Conclusion

With care, it is possible to use displacement shaders in RenderMan with no objectionable artifacts and with almost no speed penalty. The most critical parameter is the displacement bound. In most case, this parameter should be easy to calculate. Since displacement shaders operate independently of surface shaders, it is possible to displace surfaces made of any type of material.

Clearly, displacement shaders are a powerful tool for making interesting and highly detailed photorealistic images.

For more information about creating displacements refer to the **Displacements Tutorial**.

Prev | Next

Illumination

Shadows

Introduction

Adding shadows is one of the most effective ways of adding realism to a computer-generated image. The human visual system uses shadows to determine depth, light location, and direction, as well as spatial relationships between objects. Additionally, shadows are invaluable for creating moods and motivating the emotion of an image or scene.

There are several types of shadows available with Pixar's RenderMan: depth map shadows, deep shadows, or ray-traced shadows. Each method has it own advantages. Shadow maps offer efficient shadow generation and a reusable resource. Deep Shadows are a feature-rich shadow map that combine the benefits of shadow maps with support for semi-transparent and motioned-blurred shadows. Ray-traced shadows offer greater ease of use, more features, and higher quality, but at the price of slower execution speed. What follows is a discussion of these three formats.

top

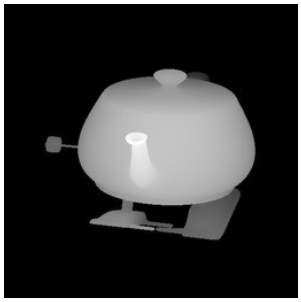
Shadow Maps

The basic idea behind shadow maps is simple. For each light that casts a shadow, a shadow map image must be rendered during a pre-pass. Each shadow map is rendered from the location of its respective light source. As an external resource, shadow maps can be reused, if light and object positions do not change. There are three types of shadow maps: depth maps, deep shadows, and soft shadows. All types generate shadow maps, but have their own features, as outlined below.

Depth Map Shadows

Overview

Depth map shadows produce good results in many situations. They are efficient but have a number of limitations (as outlined below). The basic idea is simple. For each light that casts a shadow, a shadow map image must be rendered during a pre-pass. Each shadow map is rendered from the location of its respective light source. The final result is shown in the image below. Each type of light creates distinctly characteristic shadows.



The image on the right is a depth map for the image of the teapot at the top of the page. This depth map was generated as a pre-pass before the final image was rendered. A depth map represents the distance from a specific light to the surfaces the light illuminates. A depth map contains depth info from a light's point of view. Each pixel in the depth map represents the distance from the light to the nearest shadow casting surface in a specific direction.

Limitations

Depth shadows have some limitations, which also make depth map shadows efficient by requiring less work on their part. As long as your shadows do not require the following effects, depth map shadows may suffice.

- Depth Map Limitations
  - No support for motion blur
  - No support for transparent surfaces
  - No filtering

Set Up

The simplest way to get depth map shadows with RenderMan for Maya is to create a spot, point, or directional light in your scene. RenderMan for Maya will automatically generate shadow maps when Depth Map Shadows are enabled on a light. RenderMan uses the settings in the Render Globals as defaults for the depth map, these can be overridden on a per-light basis. Here's the number of depth maps required for each light source type:

- Spot Light - 1 map
- Point Light - 6 maps
- Directional Light - 1 map

Point Light Shadow *Directional Light Shadow* *Spot Light Shadow*

Shadow Map Knobs

In the Render Globals *Passes* tab, you're able to configure the settings of your shadow maps. Here are some of the settings that are often configured on a shot by shot basis:

- *Phase* Controls how often you want to compute your shadow map. Usually you should choose "Every Frame" but if you know that the map's contents are invariant over a sequence, choose "Once Per Job."
- *Shading Rate* Controls the shading quality for your shadow map; lower numbers provide higher quality images.
- *Pixel Samples* The number of samples to take of the map. Use larger numbers when you need smooth blurring effects.
- *Ray Tracing* Ray tracing can be disabled explicitly for shadow passes.
- *Depth Filter* The algorithm for calculating shadow maps.
- *Expand checkboxes* Enables or disables Surface, Displacement, Light, and Volume shaders in the shadow map pass.

top

Deep Shadows

Overview

Deep Shadows are a feature rich depth map shadow that support transparency and volumetric data. Deep shadows can also be pre-sampled.

Deep Shadows with semi-transparent shadows

Same image rendered with Depth Map Shadows. The inferiority of the Depth Map Shadow is clear.

Deep Shadow Benefits

Deep Shadows have the following advantages over traditional depth map shadows, advantages that make deep shadows ideal for fur, or when shadowing any collection of small, semi-transparent objects.

Deep Shadow Features:

1. Semi-transparent shadows
2. Colored shadows
3. Motion-blurred shadows
4. Multi-sampling
5. Far fewer animation artifacts than standard shadow maps
6. Subtler, better-looking shadows than depth maps

Motion blurred teapot and shadows

Deep Shadow Cons

If deep shadows have a fault, it is that they store more data than standard shadow maps. This means larger files and increased render times. Deep Shadows should be used only when their benefits are useful and are not a replacement for traditional depth maps in general. Note however that by pre-filtering a deep shadow map a shadow map can be generated at a lower resolution than would be required by a traditional shadow map, which can mean that in some instances a lower resolution deep shadow map will render faster and require less disk space than an equivalent standard shadow map generated at a necessarily higher resolution.

Deep Shadows Vs. Ray Tracing

Deep Shadows can create many effects that could be also be created via ray tracing. Creating these effects with deep shadows is generally much faster and more efficient than generating the same effect with ray tracing.

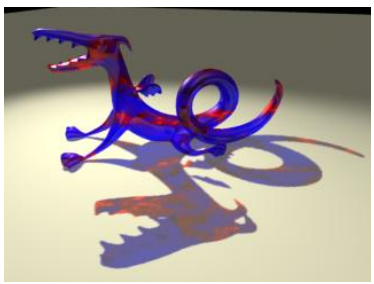
For more information see the **Deep Shadow Tutorial**.

top

Ray-Traced Shadows

Overview

Light sources can ray trace shadows, and ray-traced shadows have several features that traditional depth map shadows lack. Traced shadows can be used to create soft shadows, colored shadows, and motion-blurred shadows. They are also resolution independent, unlike shadow maps. Ray-traced shadows, however, can be costly to render.



Ray traced shadows offer these effects:

1. Semi-transparent shadows
2. Colored shadows
3. Motion-blurred shadows
4. Soft shadows

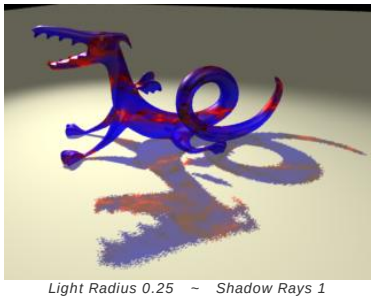
Blurred Shadows

Ray-traced shadows can create soft shadows. To create soft shadows just increase the amount of by increasing the Light Angle. The higher the value, the greater the blur (see images below). Blurred shadows usually demand a corresponding increase of the shadow "samples" ()to eliminate "dotty" artifacts.

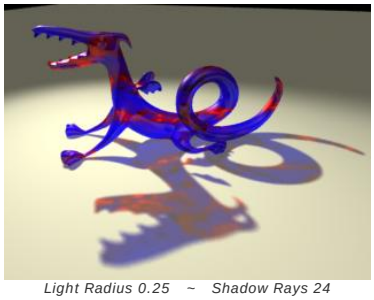
Ray-Traced Shadow Knobs

- ***Shadow Rays*** The number of rays cast. More rays allow you to achieve higher quality shadows (see images below). More rays also means slower rendering.
- ***Light Angle*** Controls the blurriness of your shadows. Increase the number of Shadow Rays to reduce noise.
- ***Bias*** The bias value for ray-traced shadows is controlled via the Trace Bias parameter in the Ray Tracing section of the Features tab in the Render Globals. Adjusting this parameter affects shadow creep. If it is too low the shadows will creep onto the front of objects. If it is too high the shadows will creep away from the bases. The size of objects, in world coordinates, affects the ideal value for this setting and must be adjusted on a per scene basis.

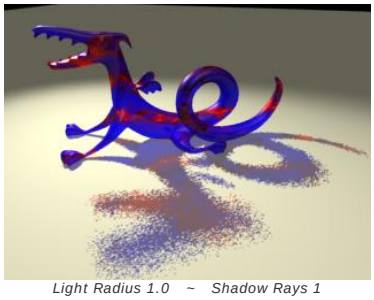
Ray-Traced Shadow Examples



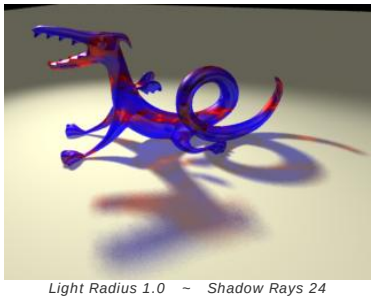
Light Radius 0.25 ~ Shadow Rays 1



Light Radius 0.25 ~ Shadow Rays 24



Light Radius 1.0 ~ Shadow Rays 1



Light Radius 1.0 ~ Shadow Rays 24

top

Prev | Next

Reflections and Refractions

Ray Tracing Reflections & Refractions

RenderMan's advanced ray-tracing sub-system allows the creation of physically correct reflections and refractions. RenderMan has special attributes available for controlling how rays behave for the creation of special effects like, blurry reflections, anti-aliasing, and bias. Below some of these controls will be explored. These controls work similarly for both reflections and refractions.

Global Controls

Ray Tracing must be enabled in the **Features Tab** of the Render Globals in order to ray trace reflections or refractions. Once ray tracing is enabled, the global controls for ray tracing are available.

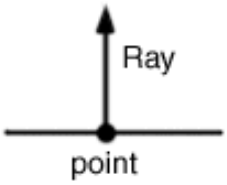
Ray Tracing Features

Adding RenderMan Controls

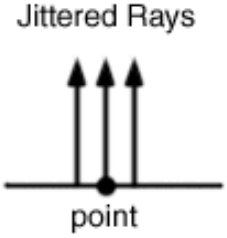
In general, RenderMan for Maya renders ray traced reflections using the same workflow as Maya. There are, however, some special ray tracing features for RenderMan which are not directly supported by Maya. You can add these controls to your Maya Materials as needed, which will allow you to take advantage of these effects. To add these controls to a Material open it in Maya's *Attribute Editor*, then go to the *Attributes* menu and select *RenderMan-> Add Reflection Controls* (or *Refraction Controls*). New ray tracing controls will appear under the *Extra RenderMan Attributes*.

Samples

By increasing the ray samples, we increase the amount of rays cast from any point. For some effects a sample of "1" may be sufficient. However, other times casting multi-sample allows for greater image quality (using multi-sampling for anti-aliasing) and special effects (like blurry reflections). Higher samples will also dramatically add more time to rendering and should be increased with care.



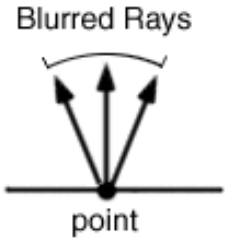
Single sample



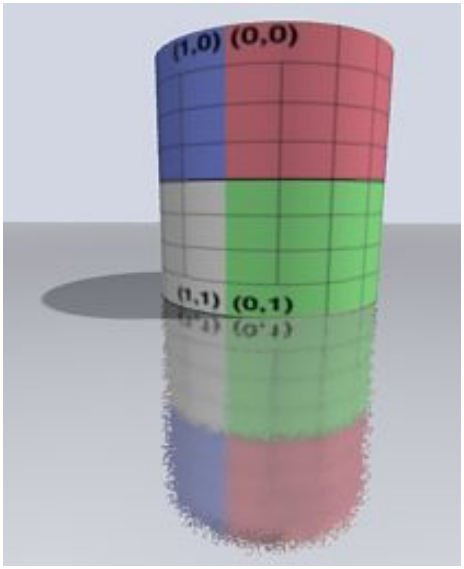
Multi-sampling

Blur

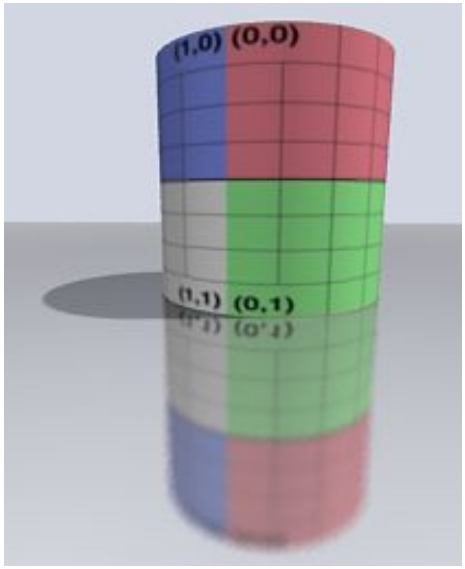
Both reflections and refractions can be blurred. Increasing the value of blur increases size of the ray's sample cone. This allows the creation of blurry reflections, refractions, subsurface scattering, etc. Blurring traces also requires a greater number of samples to be cast from any given point in order to reduce splotchy artifacts.



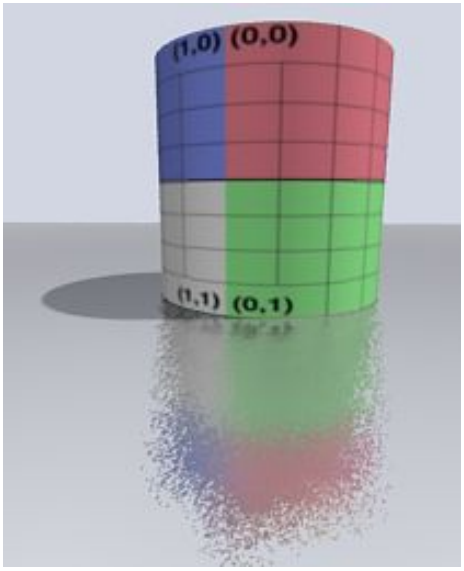
In the images on the below, the number of samples has a large effect on the quality of the resulting image. In this case, twelve samples might not be enough for a high quality render, but twelve produces a much more acceptable effect than one single sample.



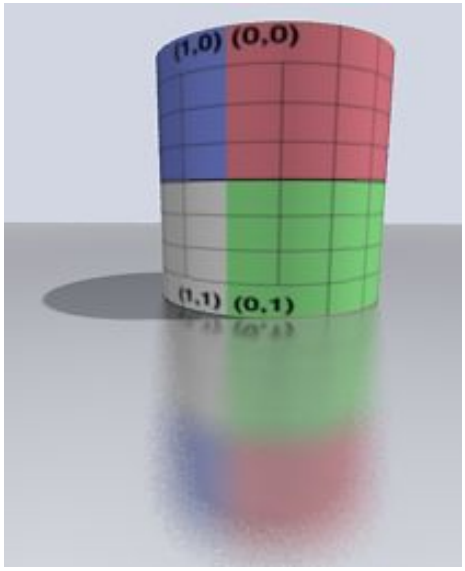
1 Sample "0.25" Blur



12 Samples "0.25" Blur



1 Sample "1.0" Blur



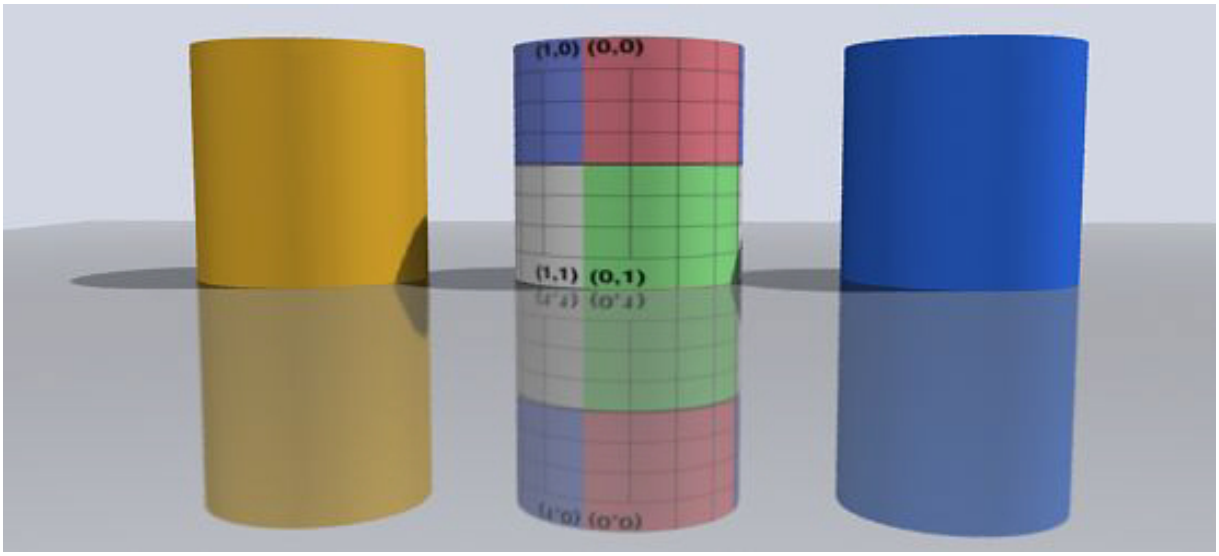
12 Samples "1.0" Blur

Trace Sets

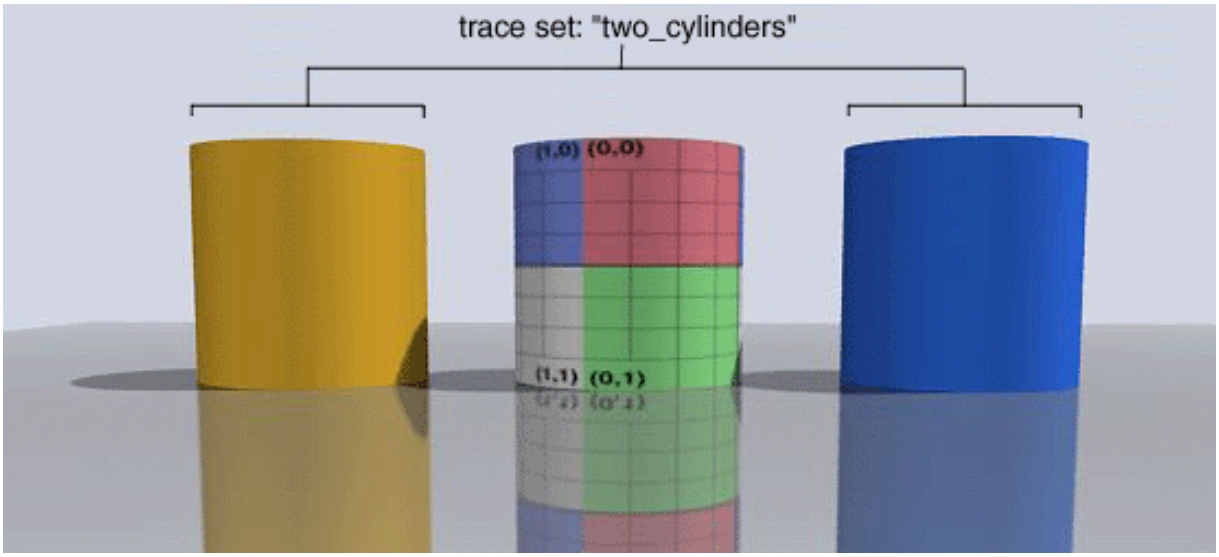
Trace sets provide a method to define which objects a particular shader will trace. Objects can be defined as trace sets (which are simple Maya sets with an added attribute). These sets can be referenced in shaders in the "Trace Set" parameter.

The visibility of objects to tracing can still be enabled and disabled on a global level (using Maya's Attribute Spread Sheet). The Trace Set brings an added level of control where objects can be visible to tracing on a per-shader basis.

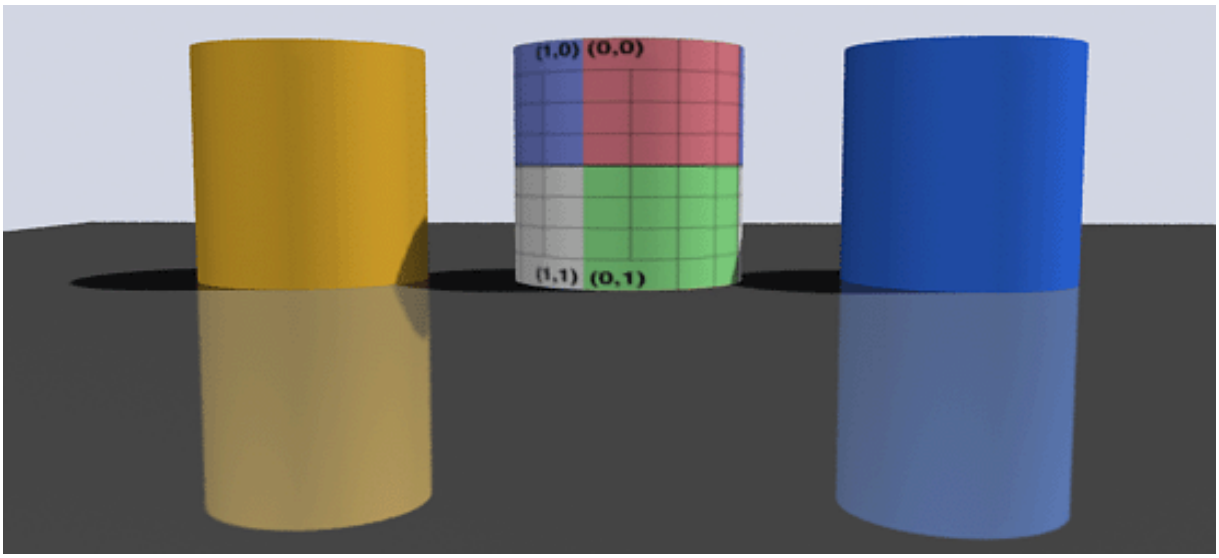
The effective use of trace sets can limit the total amount of ray tracing calculations which are performed in a scene, significantly reducing memory requirements.



Reflections without trace sets



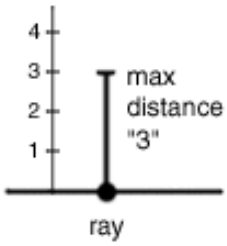
The yellow and blue cylinder are added to a trace set called, "two_cylinders"



The reflection is limited to the trace set "two_cylinders" and rendered.
Note neither the sky or the middle cylinder are included in the reflection.

Max Distance

The distance that rays travel from surfaces can be limited so that the rays are cast a fixed distance. This allows rays to sample only objects which are within a set proximity, saving the added cost of probing a scene, hitting and dicing geometry, and running shaders of remote objects that have little bearing on the effect. Max distance is especially useful when used when indirect illumination, or other effects which require many blurry samples when occlusion culling of nearby objects is most important, and far away objects have less relevance.



If a ray travels the entire max distance without hitting any objects, it expires; the ray misses. When a ray misses it returns the environment color.

Prev | Next

Global Illumination

Global Illumination

Introduction

With global illumination techniques, highly realistic lighting effects can be created with minimal setup. Ray-traced global illumination carries a large overhead, and these effects can be computationally expensive to render. However, with RenderMan for Maya 2.0 users can now take advantage of point-based approximate global illumination, a new technique, using `ptfilter`, that was introduced to RenderMan in the 13.0 release. Without taking the place of ray-traced global illumination, it offers users an alternate, faster workflow for using global illumination in their scenes.

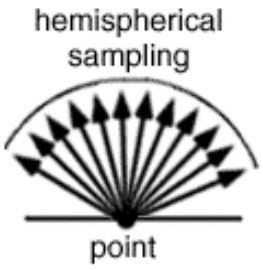
About Global Illumination

There are several methods for creating global illumination with RenderMan for Maya, but global illumination generally has two components: hemispherical sampling for shadows and image based lighting for surface illuminance. It's important to keep in mind that either component may be used on its own or both components may be used in conjuction with normal scene lighting. Next we'll explore soft shadows and image based lighting a bit more.

Soft Shadows

Hemispherical Sampling

The effect of soft shadows can be generated by determining the amount a point is obscured by other surfaces, an effect otherwise known as *occlusion*. This occlusion information can be determined by casting many hemispherical samples from a given surface point, and because so many rays must be cast, computing these calculations can be expensive.



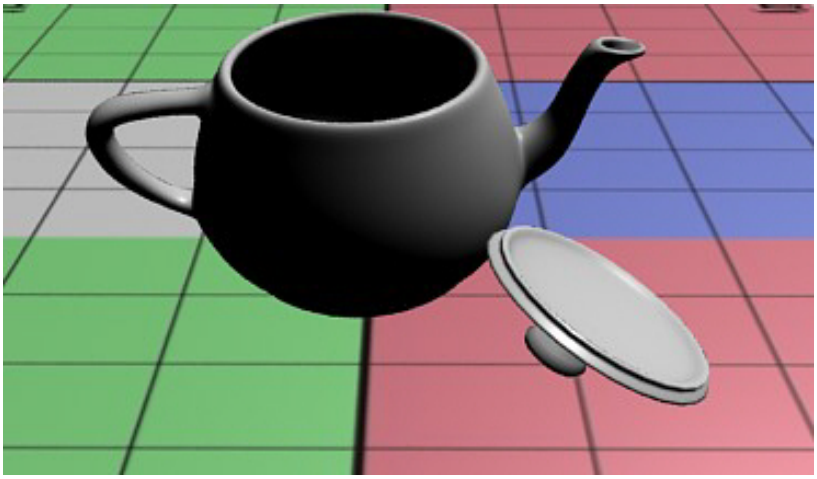
RenderMan's efficient hemispherical sampling algorithm only samples at points where it has to; and the **Max Variation** setting allows you to control the quality/speed trade off.

Global Vs. Direct Illumination

The two images below show the difference between global and direct lighting. The scene has one light, which doesn't cast shadows. The only difference between the two images is the addition of

global illumination. Global illumination provides many subtle shading effects but these effects have a high cost at render time. Subtle shadows can also be faked, and while this takes more setup time during lighting and shading, the images will render much faster. The choice to use global illumination, or to fake it, should be a conscious decision.

Direct Light



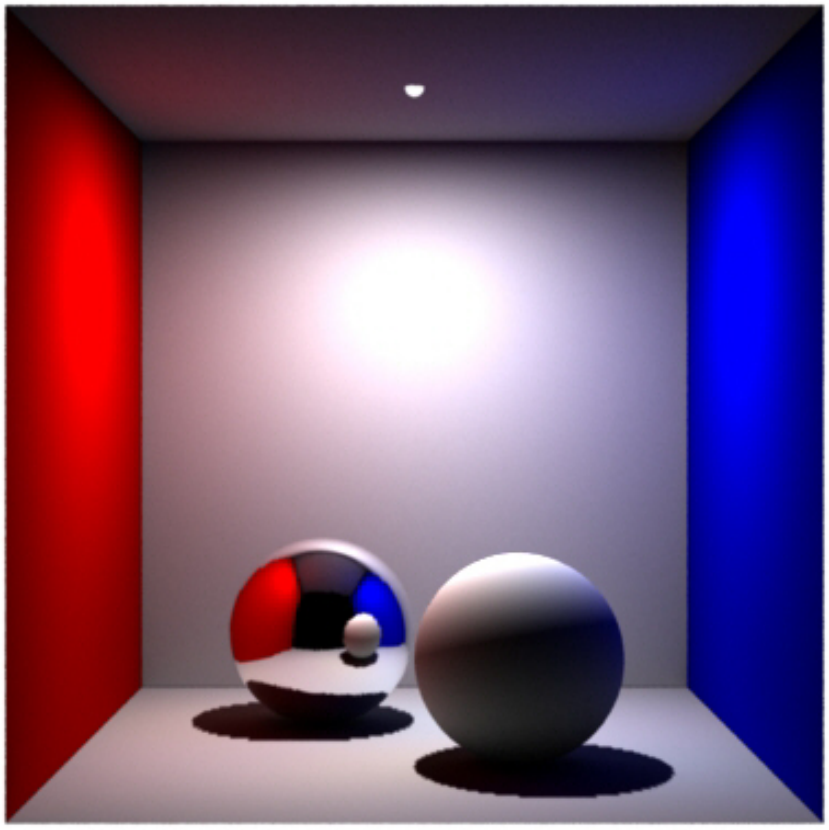
A scene with one direct light, with no shadows.

Occlusion

The same scene as above, with the addition of global illumination.

Color Bleeding

Like occlusion, color bleeding uses hemispherical sampling to calculate its effect. Color bleeding is different than occlusion because each hemispherical sample that hits a surface will run that object's shader. This is much more expensive than occlusion calculations, but it has the advantage of getting colors to bounce from one object to another. Color bleeding is also required for true image based lighting.

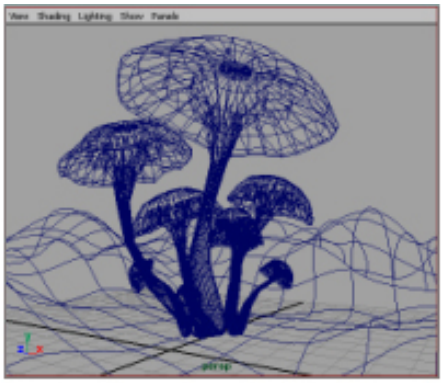


Color Bleeding
The blue color bounces off the wall onto the white sphere.

Image Based Lighting

Image based lighting uses environment maps (not scene lights) to illuminate objects. In this method scene lighting is derived from an environment map, which can be a high dynamic range image, HDRI. Through image based lighting, realistic lighting can be quickly integrated into a scene. Below is a picture of an HDRI image of a forest which will be used to illuminate the geometry on the right, the clump of mushrooms.

HDRI Image



Geometry

In the image on the right, the HDRI has been applied to the geometry using the RenderMan Environment Light. All of the lighting is coming from the image. There are no direct lights in this scene.

There are no shadows, however. We can add shadows by using the *Occlusion* shadowing mode of the RenderMan Environment Light.

Pure image based lighting

To showcase occlusion, the scene has been rendered with simple materials, without the environment map, and with occlusion enabled.

Notice how occlusion creates smooth subtle shadowing based on the amount points are obscured by other points. This shadowing will be combined with the image based lighting above.

Pure Occlusion

Finally, the scene is rendered with the complicated materials, the HDRI environment map, and with occlusion enabled.

When used together image based lighting and occlusion can produce some great effects with minimal setup.

Image based lighting with occlusion

Baking Data

Global illumination can be baked and stored on disk. Once generated, this cached data can be reused, dramatically decreasing render times.

To bake global illumination with RenderMan for Maya, create a connection to the **Bake parameter** of the RenderMan Environment Light. Once a cache file is baked it can be referenced in subsequent renders. Data can be reused as long as the relevant object(s) doesn't move. If the object moves the global illumination data must be recalculated on a per-frame basis.

Point-Based Global Illumination

Using pre-baked point clouds, RenderMan can now calculate global illumination without the expense of ray tracing. The technique is essentially an extension of baking your global illumination data. You now have the option of creating an Approximate Global Diffuse Pass connected to your RenderMan Environment Light (which can now be created without enabling ray tracing).

The advantages of our point-based approach are:

- No noise.
- Faster computation times. (No ray tracing.)
- The geometric primitives do not need to be visible for ray tracing; this can significantly reduce the memory consumed during rendering.
- Color bleeding is nearly as fast as occlusion. (No evaluation of shaders at ray hit points.)
- (HDRI) environment map illumination can be computed at the same time as the occlusion.
- Displacement mapped surfaces take no more time than non-displaced surfaces.

For More Information

The basic workflow for ray-traced and point-based global illumination is covered in the Global Illumination tutorial.

You can find more technical information about global illumination in the RenderMan Application Notes: Ambient Occlusion, Image-Based Illumination, and Global Illumination and Point-Based Approximate Ambient Occlusion and Color Bleeding.

MEL Scripting

Passes

Creating passes with MEL

RenderMan for Maya provides controls for advanced pass manipulation. Passes can be created and modified via MEL scripting. What follows is an examination of how to manipulate passes for RenderMan for Maya using MEL scripting.

How to find out what kinds of passes there are

RenderMan pass nodes in Maya are based on *nodetype* templates defined in RenderMan_for_Maya.ini. Here's what the beginning of the Shadow pass template looks like:

```
nodetype pass:render:Shadow {
```

The part of the node type after the last colon is what will be refered to as its *class*. You could browse RenderMan_for_Maya.ini looking for pass classes. An easier place to look would be under the Passes tab in the render globals. If you click on the "Create Pass" button you can see an option menu containing all the known pass classes. Another easy way to get a list of pass classes is with this MEL command:

```
rmanGetPassClasses;
// Result: Custom DeepShadow EnvCube EnvMap EnvNx EnvNy EnvNz EnvPx
EnvPy EnvPz EnvRender Final MakeGlobalDiffuse2d MakeGlobalDiffuse3d
Reference ReferenceImage Reflection RenderGlobalDiffuse2d
RenderGlobalDiffuse3d SSDiffuse SSMakeBrickmap SSRender Shadow TxMake //
```

Pass Class	Description
Custom	A basic pass that's mostly empty of settings. You'd make it into something by adding extra settings.
DeepShadow	Renders a deep shadow.
EnvCube	Makes an environment map from six input images.
EnvMap	Makes an environment map from one lat-long input image.
EnvNx	A subpass of EnvRender, renders a view down the negative x axis.
EnvNy	A subpass of EnvRender, renders a view down the negative y axis.
EnvNz	A subpass of EnvRender, renders a view down the negative z axis.

EnvPx	A subpass of EnvRender, renders a view down the positive x axis.
EnvPy	A subpass of EnvRender, renders a view down the positive y axis.
EnvPz	A subpass of EnvRender, renders a view down the positive z axis.
EnvRender	Renders six images, one down each axis, and makes an environment map out of them.
Final	A final beauty render.
MakeGlobalDiffuse2d	Generates a texture map from 2d data global illumination data. Used for baking global illumination data.
MakeGlobalDiffuse3d	Generates a brick map from point cloud data. used for baking global illumination data.
Reference	Makes a texture map out of a ReferenceImage.
ReferenceImage	Subpass of Reference which turns this image rendered by this pass into a texture map.
Reflection	
RenderGlobalDiffuse2d	A subpass of MakeGlobalDiffuse2d which takes the global illumination data rendered by this pass and makes it into a texture map.
RenderGlobalDiffuse3d	A subpass of MakeGlobalDiffuse3d which takes the 3d point cloud data rendered by this pass and makes it into a brick map.
SSDiffuse	A subpass of SSMakeBrickmap, this pass filters the point cloud generated by SSRender.
SSMakeBrickmap	This pass converts the point cloud generated by SSDiffuse into a brick map.
SSRender	This pass generates a point cloud containing subsurface scattering data.
Shadow	Renders a shadow map.
TxMake	Converts an input image file into a pixar texture

(top)

Instantiating passes

Once you know which class of pass you're interested in, instantiating it in Maya is as simple as typing the following MEL command.

Its definition looks like this:

```
global proc string rmanCreatePass(string $passclass)
```

And here's an example:

```
rmanCreatePass Shadow;
// Result: rmanShadowPass //
```

Once you've created a pass you can always find it again by looking under the Passes tab of the RenderMan Settings.

(top)

Adding an secondary output to a pass

Here's the command you can use to add an extra output to a pass. Note, passes are normally instantiated with at least one output, so adding outputs is only necessary if you want extra outputs. This command creates an output node and wires it into the pass node.

```
global proc string rmanAddOutput( string $passnode, string $channels )
```

The $channels parameter refers to the name of the output channel you're interested in.

Examples:

This causes a "specular" output to be added to all final passes.

```
rmanAddOutput rmanFinalGlobals specular;
// Result: rmanFinalOutput1 //
```

This adds an output containting surface normals to a specific render pass.

```
rmanAddOutput rmanFinalPass N;
// Result: rmanFinalOutput2 //
```

It's possible to put more than one channel into an output by separating channel names with commas. For example, this will generate an output containing u and v data.

```
rmanAddOutput rmanFinalPass "u,v";
// Result: rmanFinalOutput3 //
```

(top)

Listing known channels

There's a command called rmanGetChannelClasses which will return a string array of the channels which are understood by the rmanAddOutput command.

Example

```
rmanGetChannelClasses;
// Result: Ci Cs N Ng Oi Os P SSAlbedo SSArea SSDMFP SSRadiance
ambient diffuse environmentdir globaldiffuse incandescence irradiance
occlusion reflection refraction s shadow specular subsurface t u v
velocity //
```

(top)

How to get a list of a pass's outputs

To get a list of the output nodes of a pass, use this command:

```
global proc string[] rmanGetOutputs( string $passnode )
```

Example:

```
rmanGetOutputs rmanFinalGlobals;
// Result: rmanFinalOutputGlobals0 rmanFinalOutput1 //
```

(top)

Deleting a pass's extra outputs

You can get rid of a pass's extra outputs with this command:

```
global proc rmanDeleteOutput( string $passnode, int $idx )
```

This command takes an index, where "1" is the primary output, which can't be deleted. So the index you supply should be greater than 1.

You can get also delete extra outputs by simply deleting a pass's output nodes (which are wired in to the pass).

Example:

```
delete rmanFinalOutput1;
```

(top)

Refreshing the Attribute Editor after adding settings to a node

To cause the Attribute Editor's Extra RenderMan Attributes section to update, use this command:

```
rmanUpdateAE
```

(top)

Prev | Next

RenderMan Attributes

The Anatomy of a RenderMan attribute

All RenderMan attributes are prefixed with "rman__" (that's two underscores), and all RenderMan attributes follow this naming convention:

`rman__`*`setting-type`*`__`*`namespace`*`_`*`setting-name`*

But luckily, you don't need to remember that. There are ways to query the name of a setting which will be described in the next section.

Setting types include:

toropt

Translator options. These settings are used in the process of translating a maya scene. They apply across an entire rendering job.

torattr

Translator attributes. These settings are also used in the process of translating a maya scene. They can vary between passes, but they don't have to. For example, it's possible to render multiple passes for which only some have motionBlur enabled.

riopt

RenderMan Options are settings that affect the rendering of an entire image. The names of some riopts and riattrs may require a namespace.

riattr

RenderMan Attributes are settings that are part of the graphics state, and unlike Options can be associated with individual primitives.

param

Params can refer to either shader parameters or command parameters. In the case of shading parameters a simpler naming convention is followed: `rman__`*`param-name`*. Command parameters follow the same convention as other setting types.

(top)

How to figure out the name of an attribute

There are a couple ways you can find out the name of an attribute. The easiest is to select the node you want to add an attribute to, then open the RenderMan Attributes window (Attributes->RenderMan->Manage Attributes...) The window lists all the attributes that you can add to the selected node. These are listed as labels, because they're more readable that way. When you click on one of the labels, the corresponding name of the attribute appears in the description field at the bottom of the window. You can jot it down for future use.

Another way to find out the name of an attribute is to look in the DeclarationTable in RenderMan_for_Maya.ini. It lists all the known attributes, but their names aren't quite in the format you need. Here's are a couple example declarations:

```
declare riattr {float ShadingRate} {
    label "Shading Rate"
    range {0 100 0.1}
    subtype slider
    description "Values from 5 to 100 allow for faster
                 renders, while a value of 1 is high quality and slower."
}
declare riattr {int trace:maxspeculardepth} {
    label "Max Specular Depth"
    subtype slider
    range {0 10 1}
    description "Number of bounces for reflections and
                 refractions.  Value of 1 or 2 is sufficient unless you
                 need multi bounce effects."
}
```

The piece of this declaration that you can use to find out the corresonding attribute name in maya is the name of the setting -- the portion after the data type. In these two examples that's "ShadingRate" and "trace:maxspeculardepth". Note the latter has what's called a namespace, and the former doesn't. That's nothing you need to be particularly conscious of, just know that it takes part in the attribute name. You can pass the name of the setting to a MEL script called:

```
global proc string rmanGetAttrName( string $declname )
```

For example:

```
rmanGetAttrName "ShadingRate";
// Result: rman__riattr___ShadingRate //
rmanGetAttrName "trace:maxspeculardepth";
// Result: rman__riattr__trace_maxspeculardepth //
```

(top)

Which attributes might you want to add to a node?

To get a list of the attributes you might want to add to a node, try this command:

```
global proc string[] rmanGetOptionalAttrs( string $node )
```

Example:

```
rmanGetOptionalAttrs nurbsSphereShape1;
// Result: rman__riattr___ShadingRate rman__riattr___SmoothShade
rman__riattr___MatteObject rman__riattr___DoubleSided
rman__riattr___ReverseOrientation rman__riattr___MotionFactor
rman__riattr__cull_backfacing rman__riattr__cull_hidden
rman__riattr__derivatives_centered rman__riattr__derivatives_extrapolate
rman__riattr__dice_binary rman__riattr__dice_hair
rman__riattr__dice_rasterorient rman__riattr__grouping_membership
rman__riattr__identifier_name rman__riattr__identifier_objectid
rman__riattr__sides_backfacetolerance rman__riattr__sides_doubleshaded
rman__riattr__stitch_enable rman__riattr__stitch_newgroup
rman__riattr__trace_bias rman__riattr__trace_displacements
rman__riattr__trace_maxdiffusedepth rman__riattr__trace_maxspeculardepth
rman__riattr__trace_samplemotion rman__riattr__visibility_camera
rman__riattr__visibility_specular rman__riattr__visibility_diffuse
rman__riattr__visibility_transmission rman__riattr__visibility_midpoint
rman__riattr__visibility_photon rman__torattr___outputSurfaceShaders
rman__torattr___outputDisplacementShaders rman__torattr___outputLightShaders
rman__torattr___outputVolumeShaders rman__torattr___subdivScheme //
```

(top)

How to add an attribute to a node

Now that you know how to figure out the name of an attribute, the recommended way of adding it to a node is with a MEL script called "rmanAddAttr". Here's its definition:

```
global proc rmanAddAttr( string $node, string $attr, string $val)
```

It takes three arguments, the node name, the attribute name, and the default value. RFM will figure out the appropriate type of attribute to add to the maya node, and will convert the value string to the expected type. The value can be an empty string if you want RFM to use its own default for the setting.

Some examples:

```
rmanAddAttr nurbsSphereShape1 `rmanGetAttrName ShadingRate` "";
rmanAddAttr nurbsSphereShape1 `rmanGetAttrName "trace:maxspeculardepth"` "4";
```

(top)

How to add bunches of attributes to nodes

The selection sensitive menu entries which appear in the Attribute Editor under the Attributes->RenderMan menu usually add more than one attribute. And some of them, like "Add Subsurface Scattering", even do more complex things like creating a new network of pass nodes. It might be handy to invoke these entries via MEL rather than from the menu. Here's how you can do that.

Note, it's also possible for you to add menu entries to the Attributes->RenderMan menu. These are defined toward the end of RenderMan_for_Maya.ini.

The command definition is:

```
global proc rmanExecAEMenuCmd( string $node, string $menuItemLabel)
```

Here's an example which adds subsurface scattering to a material node:

```
rmanExecAEMenuCmd blinn1 "Add Subsurface Scattering";
```

(top)

How to delete attributes

There's nothing special you need to know about deleting attributes. It's done in the same way as deleteing other maya attributes, with the deleteAttr command.

Example:

```
deleteAttr nurbsSphereShape1.rman__riattr___ShadingRate
```

(top)

How to set attribute values

If you know the name of the node and attribute, you can use maya's setAttr command. RFM also provides a command that might make setting attributes a little easier in the respect that you don't need to know the data type of the attribute, and the value can be supplied as a string. Feel free to use whichever you prefer. This is its definition:

```
 global proc rmanSetAttr(string $node, string $attr, string $val)
```

And some examples:

```
rmanSetAttr nurbsSphereShape1 rman__riattr___ShadingRate 5;
setAttr nurbsSphereShape1.rman__riattr___ShadingRate 5;

rmanSetAttr renderManGlobals rman__riopt___PixelSamples "5 5";
setAttr renderManGlobals.rman__riopt___PixelSamples 5 5;

rmanSetAttr renderManGlobals rman__GDScheme occlusion;
setAttr -type "string" renderManGlobals.rman__GDSchemeocclusion;
```

(top)

How to set Render Global values

The render globals nodes may not exist in your maya scene; they're typicaly created when the Render Globals window is raised. This command can be used to create them:

```
rmanCreateGlobals;
```

This command creates an environment light:

```
rmanCreateEnvLight;
```

Render globals can be set like other attributes with the `rmanSetAttr` command; the trick is knowing which attribute each render global corresponds to. Here's a table...

Render Global	**node**	**attribute**	**type**	**values**
Common				
Image File Output				
File Name Prefix	defaultRenderGlobals	imageFilePrefix		
Frame/ Animation Ext	rmanFinalGlobals	rman__torattr___passNameFormat rman__torattr___passExtFormat		
Image Format	rmanFinalOutputGlobals0	rman__riopt__Display_type		
Start Frame	defaultRenderGlobals	startFrame		
End Frame	defaultRenderGlobals	endFrame		
By Frame	defaultRenderGlobals	byFrameStep		
Frame Padding	defaultRenderGlobals	extensionPadding		
Renderable Objects	defaultRenderGlobals	renderAll	int	0 1
Camera	*camera node*	renderable		
RGB Channel (Color)	*camera node*	image		
Alpha Channel (Mask)	*camera node*	mask		
Depth Channel (Z Depth)	*camera node*	depth		
Custom Extension				
Use Custom Extension	defaultRenderGlobals	outFormatControl		
Renumber Frames				
Renumber Frames Using	renderManGlobals	rman__toropt___renumber		
Start Frame	renderManGlobals	rman__toropt___renumberStart		

By Frame	renderManGlobals	rman__toropt___renumberBy		
Resolution				
Maintain Width/Height Ratio	defaultRenderGlobals	aspect		
Width	rmanFinalGlobals	rman__riopt__Format_resolution0		
Height	renderManGlobals	rman__riopt__Format_resolution1		
Pixel Aspect Ratio	renderManGlobals	rman__riopt__Format_pixelaspectratio		
Enable Default Light	defaultRenderGlobals	enableDefaultLight		
Pre Render MEL	rmanFinalGlobals	rman__torattr___preRenderScript		
Post Render MEL	rmanFinalGlobals	rman__torattr___postRenderScript		
Quality				
Shading Rate	renderManGlobals	rman__riattr___ShadingRate	float	
Pixel Samples	renderManGlobals	rman__riopt___PixelSamples	float [2]	
Filter	rmanFinalOutputGlobals0	rman__riopt_Display_filter	string	box triangle catmull-rom sinc gaussian mitchell seperable-catmull-rom blackman-harris
Filter Size	rmanFinalOutputGlobals0	rman__riopt_Display_filterwidth	float [2]	
Features				
Motion Blur				
Motion Blur	renderManGlobals	rman__torattr___motionBlur	int	0 1
Camera Blur	renderManGlobals	rman__torattr___cameraBlur	int	0 1
Shutter Angle	renderManGlobals	rman__torattr___shutterAngle	float	
Shutter Timing	renderManGlobals	rman__toropt___shutterTiming	string	frameOpen frameCenter frameClose
Motion Blur Type	renderManGlobals	rman__toropt___motionBlurType	string	frame subframe
Ray Traced Motion Blur	renderManGlobals	rman__riattr__trace_samplemotion	int	0 1
Ray Tracing				
Ray Tracing	renderManGlobals	rman__torattr___rayTracing	int	0 1
Trace Bias	renderManGlobals	rman__riattr__trace_bias	float	

Max Ray Depth	renderManGlobals	rman__riopt__trace_maxdepth	int	
Max Specular Depth	renderManGlobals	rman__riattr__trace_maxspeculardepth	int	
Max Diffuse Depth	renderManGlobals	rman__riattr__trace_maxdiffusedepth	int	
Environment Light				
Environment Image	RenderManEnvLightShape1	rman__EnvMap	string	occlusion irradiance
Environment Color	RenderManEnvLightShape1	rman__EnvColor	float [3]	
Intensity	RenderManEnvLightShape1	rman__EnvStrength	float	
Emit Specular	RenderManEnvLightShape1	rman__EnvEmitSpecular	int	0 1
Emit Diffuse	RenderManEnvLightShape1	rman__EnvEmitDiffuse	int	0 1
Primary Visibility	RenderManEnvLightShape1	rman__LightPrimaryVisibility	int	0 1
Shadowing	RenderManEnvLightShape1	rman__EnvGIScheme	string	none occlusion colorbleeding
Shadow Bias	RenderManEnvLightShape1	rman__EnvShadowBias	float	
Shadow Gain	RenderManEnvLightShape1	rman__EnvShadowGain	float	
Occlusion Color	RenderManEnvLightShape1	rman__EnvOcclusionColor	float [3]	
Sampling Mode	RenderManEnvLightShape1	rman__EnvMapScheme	string	filtered sampled baked
Samples	RenderManEnvLightShape1	rman__EnvSamples	float	
Max Variation	RenderManEnvLightShape1	rman__EnvGIMaxVariation	float	
Diffuse Softness	RenderManEnvLightShape1	rman__EnvGIHemisphere	float	
Max Dist	RenderManEnvLightShape1	rman__EnvGIMaxDist	float	
Subset	RenderManEnvLightShape1	rman__EnvGISubset	string	
Color Correct	RenderManEnvLightShape1	rman__EnvColorCorrect	int	0 1
Saturation	RenderManEnvLightShape1	rman__EnvColorSaturation	float	
Bias	RenderManEnvLightShape1	rman__EnvColorBias	float [3]	
Gain	RenderManEnvLightShape1	rman__EnvColorGain	float [3]	
Bake	RenderManEnvLightShape1	rman__GDMap	string	
Passes				
Enable Render Layers	renderManGlobals	rman__toropt___enableRenderLayers	int	0 1
DeepShadow Globals				

Phase	rmanDeepShadowGlobals	rman__torattr___phase		
Resulotion	rmanDeepShadowGlobals	rman__riopt__Format_resolution	int[2]	
Shading Rate	rmanDeepShadowGlobals	rman__riattr___ShadingRate	float	
Pixel Samples	rmanDeepShadowGlobals	rman__riopt___PixelSamples	float [2]	
Motion Blur	rmanDeepShadowGlobals	rman__torattr___motionBlur	int	0 1
Ray Tracing	rmanDeepShadowGlobals	rman__torattr___rayTracing	int	0 1
Expand Surface Shaders	rmanDeepShadowGlobals	rman__torattr___outputSurfaceShaders	int	0 1
Expand Displacement Shaders	rmanDeepShadowGlobals	rman__torattr___outputDisplacementShaders	int	0 1
Expand Light Shaders	rmanDeepShadowGlobals	rman__torattr___outputLightShaders	int	0 1
Expand Volume Shaders	rmanDeepShadowGlobals	rman__torattr___outputVolumeShaders	int	0 1
Final Globals				
Image Format	rmanFinalOutputGlobals0	rman__riopt__Display_type	string	
Channels	rmanFinalOutputGlobals0	rman__riopt__Display_mode	string	
Filter	rmanFinalOutputGlobals0	rman__riopt__Display_filter	string	
Filter Size	rmanFinalOutputGlobals0	rman__riopt__Display_filterwidth	float [2]	
Exposure	rmanFinalOutputGlobals0	rman__riopt__Display_exposure	float [2]	
Shadow Globals				
Phase	rmanShadowGlobals	rman__torattr___phase	string	/Job/ Preflight/ Maps/ Shadow /Job/Frames/ Maps/ Shadow
Resulotion	rmanShadowGlobals	rman__riopt__Format_resolution	int[2]	
Shading Rate	rmanShadowGlobals	rman__riattr___ShadingRate	float	
Pixel Samples	rmanShadowGlobals	rman__riopt___PixelSamples	float [2]	
Ray Tracing	rmanShadowGlobals	rman__torattr___rayTracing	int	0 1
Default Surface Shaders	rmanShadowGlobals	rman__torattr___defaultSurfaceShader	string	
Depth Filter	rmanShadowGlobals	rman__riopt__Hider_depthfilter	string	

Expand Displacement Shaders	rmanShadowGlobals	rman__torattr___outputDisplacementShaders	int	0 1
Expand Light Shaders	rmanShadowGlobals	rman__torattr___outputLightShaders	int	0 1
Expand Volume Shaders	rmanShadowGlobals	rman__torattr___outputVolumeShaders	int	0 1
Depth Filter	rmanShadowGlobals	rman__riopt__Hider_depthfilter	string	min max average midpoint
Advanced				
Render Options				
Output Statistics	renderManGlobals	rman__torattr___outputStatistics	int	
Bucket Size	renderManGlobals	rman__riopt__limits_bucketsize	int[2]	
Grid Size	renderManGlobals	rman__riopt__limits_gridsize	int	
Extreme Displacement	renderManGlobals	rman__riopt__limits_extremedisplacement	int	
Z Threshold	renderManGlobals	rman__riopt__limits_zthreshold	float [3]	
O Threshold	renderManGlobals	rman__riopt__limits_othreshold	float [3]	
Volume Shading Rate	renderManGlobals	rman__riopt__limits_vprelativeshadingrate	float	
Reference Frame	renderManGlobals	rman__torattr___referenceFrame	int	
Output Directories				
Final Images	renderManGlobals	rman__toropt___imageOutputLoc	string	
Texture Cache	renderManGlobals	rman__toropt___textureOutputLoc	string	
Shaders	renderManGlobals	rman__toropt___shaderOutputLoc	string	
Render Data	renderManGlobals	rman__toropt___renderDataOutputLoc	string	
Cleanup				
Job Cleanup	renderManGlobals	rman__toropt___jobCleanupPattern	string	None Shadows All
Frame Cleanup	renderManGlobals	rman__toropt___frameCleanupPattern	string	None Shadows All

---

Tip

Another place to look to figure out attribute names is in the mouse-over tooltip for each control in the Render Globals window.

RenderMan for Maya Pro

RenderMan Studio includes a Pro version of the RenderMan for Maya plugin. This enhanced plugin introduces support for advanced rendering features, unleashing the full power of RenderMan Pro Server while maintaining the ease of use and seamless integration of RenderMan for Maya. Follow the links below or in the navigation at left to learn more about RenderMan for Maya's Pro features.

RfM Pro, RIB, and You

Important

These features are supported by the RenderMan for Maya Pro plugin in RenderMan Studio and are not available with the regular RenderMan for Maya plugin.

Introduction

Perhaps the biggest difference in RenderMan for Maya Pro in RenderMan Studio is the addition of support for RIB, bringing together the simplicity of the RfM workflow with the flexibility and power of a RIB-based pipeline. In addition to basic ribgen (fully archived for efficiency, by the way), there is support for RIB boxes using per-node MEL commands (including Ri commands via MEL), lazy ribgen, and dynamic RIB archives, users can manually export RIB archives, and there is a new RIB option for Maya command line renders, `Render -r rib`, which opens still more options for distributed renders. Not to mention, RIB can be generated for selected geometry, for a given render layer, for an implicit passs of a given camera or light, for an explicit pass, or for a class of passes. How cool is that?

Spooling RIB

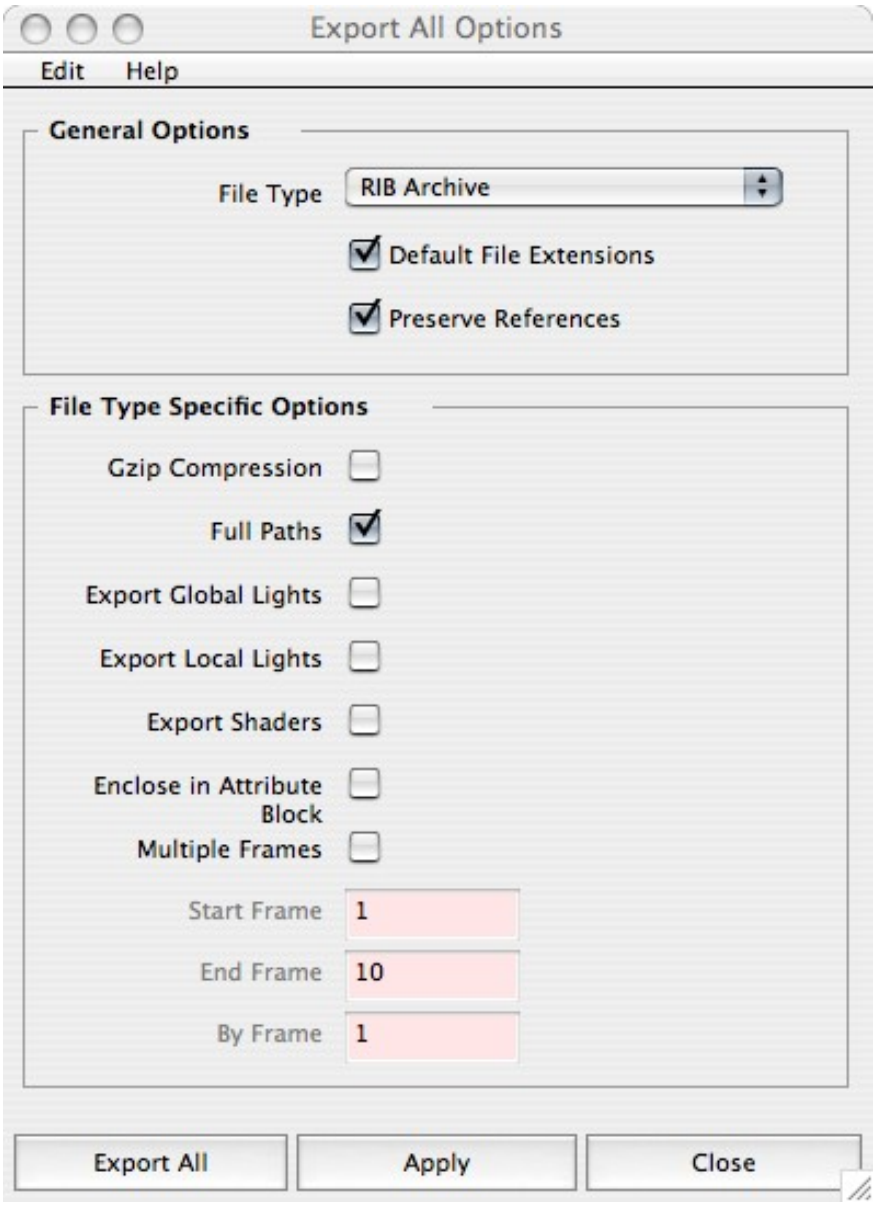
Spooling RIB is just about the easiest thing you can do. When you've selected RenderMan as your renderer in Maya, the Batch Render options menu offers you a simple interface with **Alfred**, the work distribution software included with RenderMan Studio. Once you select Alfred Spool you can do immediate, deferred, or remote RIB generation and select **netrender** (with immediate ribgen) or **prman** (with deferred or remote ribgen) via Alfred and RenderMan Pro Server. You can also choose remote or local Maya Batch renders via RenderMan for Maya.

Exporting RIB

If spooling RIB is just about the easiest thing you can do, exporting RIB pretty much *is* the easiest thing you can do. It's all handled simply through Maya's Export function, and you can export RIB archives of your entire scene or any given selection.

To export your RIB archive:

1. Go to **File -> Export All / Selection** and click on the *Options* button.
2. Select RIB Archive from the dropdown list. You will be presented with *File Type Specific Options* to choose from, as seen in the image below:



Choose your options wisely. Note that you can choose to export (or not) lights and shaders, mutliple or single frames, and you can apply gzip compression to your RIB archive (quite handy indeed).

3. Click the *Export* button. Make sure you browse to a convenient location to use to save your exported RIB archives.

In addition, you also generate RIB via a simple MEL command: `rman genrib`. Boom! RIB generated.

Command Line RIB Options

Outside of the Maya UI, you still have a world of options for generating RIB from your Maya scene files. Using Maya's **Render** command via a command prompt or terminal window. Users familiar with using Render via the command line will recognize the `-r` option it is used to select a particular renderer, such as the Maya software renderer (`-r sw`), or RenderMan (`-r rman`). Now you can render directly to RIB using the `-r rib` option:

```
Render -r rib scene.ma
```

Here is a list of additional options supported with `-r rib`:

- **All purpose flags:**

-setAttr string string

This flag can be used to set any of the global attributes listed in RenderMan_for_Maya.ini. It takes a name value pair. Attribute values which have multiple data elements should be surrounded by quotes. The flag can be used multiple times. Example:

```
Render -r rman -setAttr ShadingRate 5 -setAttr PixelSamples "3 3" -setAttr
motionBlur 1 -setAttr Format:resolution "320 240" filename
```

-setPref string string

This flag can be used to set any of the preferences listed in `RenderMan_for_Maya.ini`. It takes a name value pair. Attribute values that have multiple data elements should be surrounded by quotes. The flag can be used multiple times. For example:

```
Render -r rman -setPref BatchCompileMode zealous filename
```

- **General purpose flags:**

-rd path

Directory in which to store image files

-fnc string

File Name Convention:

```
name, name.ext, name.#.ext, name.ext.# name.#, name#.ext, name_#.ext
```

As a shortcut, numbers 1, 2, 3 can be used.

-im filename

Image file output name

-of string

File format of output images: Alias, Cineon, MayaIFF, OpenEXR, SGI8, SGI16, SoftImage, Targa, Tiff8, Tiff16, Tiff32, Iceman

- **Frame numbering options:**

-s float

Starting frame for a sequence

-e float

End frame for a sequence

-b float

By frame/step for a sequence

-pad int

Number of digits in the frame number included in the output image file name

-rfs int

The initial (renumbered) frame number for the first frame when rendering

-rfb int

The step by which frames are renumbered (used in conjunction with `-rfs`).

- **Camera options:**

-cam name

The name of the camera from which you are rendering

-rgb boolean

Enable/disable RGB output

-alpha boolean

Enable/disable Alpha output

-depth boolean

Enable/disable Depth output

-iip

Disable all image planes before rendering

-res int int

Specify the resolution (X Y) of the rendered image

-crop float float float float

Specify a crop window for the rendered image

- **Render Layers:**

-rl boolean|name(s)

Render each listed layer separately

- **MEL callbacks:**

-pre string

MEL code executed before each frame

-post string

MEL code executed after each frame

  - **MEL callbacks for Maya 7.0**

-preRender string

MEL code executed before rendering

-postRender string

MEL code executed after rendering

-preLayer string

MEL code executed before each render layer

-postLayer string

MEL code executed after each render layer

-preFrame string

MEL code executed before each frame

-postFrame string

MEL code executed after each frame

- **Bake Options:**

-bake int

  - **0**: Don't bake, but do regular rendering
  - **1**: Bake texture maps
  - **2**: Bake texture maps and do regular rendering

-bakeChannels string

Comma delimited list of one or more channels: _ambient,_diffuse,_diffuse_noshadow, _incandescence,_indirect,_indirectdiffuse, _irradiance,_occlusion,_reflection,_refraction, _shadow, _specular,_subsurface,_surfacecolor, _translucence

-bakeResolution int int

Set X Y resolution of baked maps

-bakeCamera string

Camera to use while baking

-bakeFileFormat string

File format of output images: Alias, Cineon, It, MayaIFF, OpenEXR, SGI8, SGI16, SoftImage, Targa, Tiff8, Tiff16, Tiff32

-bakeFileDepth string

Depth of output images: byte, short, float

- **Spool Options:**

-spool string

```
"mayabatch local"
"mayabatch remote"
"immediate rib, local render"
"deferred rib, remote render"
"remote rib, remote render"
```

-alfredui

Launch Alfred with its User Interface

-chunkSize

`-chunkSize` sets the Frames Per Server. This option is only applicable when used in conjunction with the `-spool` option.

- **Other:**

-rep boolean

Do not replace the rendered image if it already exists

-n int

Number of processors to use. `0` indicates use all available.

-compile boolean

Forces compilation of all shaders, even if they already exist.

Important

- Remember to place a space between option flags and their arguments.
- Any boolean flag will take the following values as TRUE: on, yes, true, or 1.
- Any boolean flag will take the following values as FALSE: off, no, false, or 0.

Additionally, a complete list of the options can also be seen by running the `Render -r rman -h` command.

More Information

For more information about RIB in general, it's not a bad idea to check out the RenderMan Interface documentation in the RenderMan Manual.

For more information about spooling renders via Alfred, please consult the Alfred documentation.

For a tutorial on some simple RIB archiving, check out the Dynamic Read Archives tutorial.

Advanced Usage of RSL in RfM Pro

Important

These features are supported by the RenderMan for Maya Pro plugin in RenderMan Studio and are not available with the regular RenderMan for Maya plugin.

RenderMan for Maya has always offered support for using the RenderMan Shading Language, to a certain degree. Hand-written, compiled shaders can be imported as a *RenderMan Shader* node in Maya. RenderMan for Maya Pro introduces a new way to access the awesome power of RSL: **RSLFunctionNodes**.

RSLFunctionNodes appear to the enduser as plugin Maya nodes that can be "wired" into standard Maya shading networks for use with RfM. They are radically simpler to create than a standard Maya plugin node because RfM handles all the details associated with registering and populating Maya nodes with attributes. RSLFunctionNodes are described in a manner patterned after Slim Templates. There is an association between each RSLFunctIonNode type and RenderMan Shading Language (RSL) functions. End-users can trivially extend the collection of RSLFunctionNodes and even override those provided by Pixar.

Ri for MEL

Important

These features are supported by the RenderMan for Maya Pro plugin in RenderMan Studio and are not available with the regular RenderMan for Maya plugin.

RenderMan for Maya has superb support for RenderMan Attribute and Option control based on Maya attributes. This data-centric support is fundamental to Maya's operation and offers advantages for file reference parameter animation and general orthogonality with Maya tools.

There are times where *procedural control* of RenderMan state is desirable. We offer this capability in RfM 2 Pro through a MEL binding to RenderMan Interface procedures. Whether rendering directly (internally) or to produce RIB, these new commands can be used to tailor the Ri stream via mel procedures. New *torattrs* are supported to provide hooks for custom mel procedures. These can be assigned to shape, light and transform nodes and will be executed when RfM is rendering.

`defaultRiOptionsScript`

Runs during rendering/ribgen and provides a hook to override rendering options via script control.

`defaultRiAttributesScript`

Runs during rendering/ribgen and provides a hook to override default rendering attributes via script control

`transformBeginScript`

Runs during rendering/ribgen at each transform nodeimmediately following RiAttributeBegin.

`postTransformScript`

Runs during rendering/ribgen at each transform node immediately following the setup of the transformation state.

`transformEndScript`

Runs during rendering/ribgen at each transform nodeimmediately preceeding RiAttributeEnd.

`preShapeScript`

Runs during rendering/ribgen at each shape node immediatelypreceeding geometry output.

`postShapeScript`

runs during rendering/ribgen at each shape node immediately following geometry output.

Ri Commands

Syntax

Ri commands fall into two broad syntactic classes: commands with and commands without variable length parameter lists. Those commands without variable length parameter lists generally accept typed, positional arguments following the RIB binding conventions. For example, a red color is specified this way:

```
RiColor 1.0 0.0 0.0
```

Since Ri parameter lists are comprised of a heterogeneity of types, we follow the Ri-convention that requires parameter-type declarations be provided. Modern practice combines the type declaration with the parameter name to produce an *inline declaration*.

Inline Declaration Syntax

```
"type name" ''value''
"type[arraylen] name" ''value1'' ... ''valueArrayLen''
```

Parameterlist Examples

```
RiAttribute "identifier" "string name" "myname"
RiAttribute "visibility" "int camera" 1 "int trace" 0
RiSurface "mySurfaceShader" "string[2] myFiles" "file one" "file two"
           "float Ks" 1.0 "float[5] myweights" 1 2 3 4 5;
```

Scope

RiBegin *filename_string*
RiEnd
RiFrameBegin *frame_int*
RiFrameEnd
RiWorldBegin
RiWorldEnd
RiAttributeBegin
RiAttributeEnd
RiTransformBegin
RiTransformEnd
RiSolidBegin *operation_string* ("primitive", "intersection", "union", "difference")
RiSolidEnd
RiResourceBegin
RiResourceEnd

Options

RiOption *namespace_string ...parameterlist...*
RiHider *hidername_string ...parameterlist...*

-

RiFormat *xres_int yres_int pixelaspectratio_float*
RiPixelSamples *xsamples_int ysamples_int*
RiScreenWindow *left_float right_float bottom_float top_float*
RiCropWindow *xmin_float xmax_float ymin_float ymax_float*
RiProjection *projtype_string ...parameterlist...*
RiClipping *near_float far_float*
RiDepthOfField *fstop_float focallength_float focaldistance_float*
RiShutter *min_float max_float*

-

RiDisplayChannel *channeldecl_string ...parameterlist...*
RiDisplay *name_string type_string mode_string ...parameterlist...*

-

RiRelativeDetail *relativedetail_float*

Attributes

RiAttribute *namespace_string ...parameterlist...*
RiColor *red_float green_float blue_float*
RiOpacity *red_float green_float blue_float*

-

RiSurface *shadername_string ...parameterlist..*
RiDisplacement *shadername_string ...parameterlist..*
RiAtmosphere *shadername_string ...parameterlist..*
RiInterior *shadername_string ...parameterlist..*
RiExterior *shadername_string ...parameterlist..*

-

RiLightSource *shadername_string lighthandle_string ...parameterlist...*
RiIlluminate *lighthandle_string onoff_bool*

-

RiShadingRate *size_float*
RiGeometricApproximation *type_string value_float*

-

RiShadingInterpolation *interp_string* (smooth, constant)
RiMatte *onoff_bool*

-

RiDetail *minx maxx miny maxy minz maxy* (floats)
RiDetailRange *minvisible lowertransition uppertransition maxvisible* (floats)

-

RiSides *sides_int* (1 or 2)
RiOrientation *orientation_string* (lh, rh, inside, outside)
RiReverseOrientation

-

RiResource *handle_string type_string ...parameterlist...*

Transform

RiIdentity
RiRotate *angle dx dy dz* (floats)
RiScale *sx sy sz* (floats)
RiSkew *angle dx1 dy1 dz1 dx2 dy2 dz2* (floats)
RiTranslate *x y z* (floats)
RiCoordinateSystem *coordsysname_string*
RiScopedCoordinateSystem *coordsysname_string*

Geometry

RiSphere *radius zmin zmax angle ...parameterlist...*

-

RiProcedural (work only with RIB out)
RiProcedural "DelayedReadArchive" *filename xmin xmax ymin ymax zmin zmax*
RiProcedural "DynamicLoad" *dsoname xmin xmax ymin ymax zmin zmax* ...paramlist...

RenderMan for Maya Plugins

Ri Filters (Rifs) in RfM Pro

Important

These features are supported by the RenderMan for Maya Pro plugin in RenderMan Studio and are not available with the regular RenderMan for Maya plugin.

RenderMan for Maya Pro , as part of RenderMan Studio, has a new User Interface for accessing Ri Filters (Rifs), under the Advanced tab. The UI displays a list of Rifs and allows you to add, delete, or rearrange the list. Rifs are applied globally to all rendering passes, but there's a checkbox for globally enabling or disabling Rifs.

Rifs are represented as RenderMan settings nodes, much like displays and passes. When you add a Rif, a new node is created and wired into a message array attribute on the renderManGlobals node. The order of connections to the array indicates the order in which Rifs will be applied, and the connections are taken care of via the UI.

An example is forthcoming.

RiProcedurals in RfM Pro

Important

These features are supported by the RenderMan for Maya Pro plugin in RenderMan Studio and are not available with the regular RenderMan for Maya plugin.

RenderMan for Maya Pro supports procedurals simply by creating a MEL script attribute either on a shared geometric attributes node or directly on a transform or shape node.

You can find simple examples of using procedurals with RenderMan for Maya in the Tutorials section of the documentation. Please see the tutorials for Dynamic Read Archives and Custom Geometry.

What is Supported

What is Supported?

• Important

RenderMan for Maya translates the majority of Maya elements including, but hardly limited to:

CAMERAS
Cameras (No stereographic rendering)
ImagePlane
GEOMETRY
NURBS (True curved surfaces. No tessellation issues)
Polygons
Hierarchical Subdivision Surfaces (True curved surfaces. No tessellation issues)
PARTICLES
Streak
Multistreak
Point
Multipoint
Sphere
Cloud
Sprite
Blobby Surface
Particle Instancing
LIGHTS
Ambient
Directional
Point
Spot
Area
Volume
MAYA MATERIALS
SurfaceShader
VolumeShader
DisplacementShader
Material nodes unsupported by RenderMan for Maya are listed **here**.
MAYA UNLIMITED
Fur (Support for majority of attributes, including keyframe animation & dynamics)
Hair (Requires Maya Unlimited 6.5 or higher)
Cloth
Paint Effects (Partial implementation; Requires Maya 6.5 or higher)

Known Limitations

Unsupported Maya Features

GLOW - Maya glow (and other post-process effects) are not supported.
IPR -Maya's interactive renderer is not supported by RenderMan for Maya.
FLUID EFFECTS -Not supported at this time.

Unsupported Maya Materials

Surface Materials
Ocean Shader
Shading Map
Volumetric Materials
Env Fog
Fluid Shape
Volume Shader

2D Textures
Fluid Texture 2D
Movie
Ocean
PSD File
Water
3D Textures
Fluid Texture 3D
Env Textures
Env Cube
Env Sky
General Utilities
Stencil
Distance Between
(tbd)
Color Utilities
(tbd)
Switch Utilities
baseShadingSwitch
double

Particle Utilities
particleColorMapper
IncandMapper
TranspMapper
AgeMapper
(tbd)

Partially Supported Materials
particleCloud (work-in-progress)
cloud (Issues with softedges)
leather (Close match)
granite (Close match)
light fog (Close match, extra sampling features)
particleSampler (Partial Implementation)
Env Ball (support for HDR environment probes, eyespace and projection geometry are broken)