`

Autodesk® Maya®

Render Pass Concepts and Techniques

A rendering guru's guide to harnessing the power of Maya

Who Should Read this Document?

This whitepaper is meant to supplement the Autodesk® Maya® software User Guide for versions 2009 through 2011. It is not intended to be an exhaustive reference, and it will not necessarily be updated at each release. The purpose of this document is to provide information that is beyond the scope of the User Guide on how render passes work and how they can be used. The target audience is Maya generalists, rendering specialists, technical directors specialized in rendering, production pipeline engineers, rendering plug-in developers, and compositing specialists. Unless otherwise stated, all the functionality described in this document pertains to mental ray® rendering software, versions 3.7 and later, as integrated in the Autodesk Maya software. The specific integrations provided for/by other renderers may vary.

It should also be noted that this document does not provide user-interface level instructions on how to use render passes in Autodesk Maya. That is covered in the Maya User Guide.

It is assumed that the reader is an intermediate to expert Maya and mental ray user with a good understanding of fundamental rendering, ray-tracing, and compositing concepts and techniques.

Autodesk®

Contents

1. What Are Maya Render Passes?

In cinematography, the term *pass* traditionally refers to one of many geometrically coherent shots taken in motion control photography, which are then optically or digitally composited to form a visual effects shot. Render passes in Autodesk® Maya® software are a metaphor for an analogous computer graphics process in which several coherent shots are produced, and subsequently combined using image compositing tools. However, the objectives of the two are completely different: motion control passes help solve the problem of combining images of objects that cannot be filmed simultaneously, such as multiple copies of the same actor, or physical objects that are at different scales; on the other hand, Maya render passes are designed to deliberately decompose a renderable scene into multiple component images that can be altered independently before recombining them in compositing. Illustrated examples of render pass compositing techniques are given in section 14 of this document.

The Maya notion of a render pass is not to be confused with the passes used in multi-pass rendering techniques, which usually consist of rendering the same scene multiple times to achieve effects that cannot be computed conveniently or efficiently by rendering the scene only once. Examples of such techniques include glow effects, reflection maps, shadow maps, and accumulation buffer-based effects (e.g. some motion blur and depth of field techniques). By contrast, Maya render passes are typically rendered simultaneously, which helps make it a computationally efficient process.

1.1. Old *vs.* New Render Passes

In the Maya User Guide, the term *multi-render pass* is used to refer to the render pass technology introduced in Maya 2009, which is the object of this document. That term is used in the guide to avoid confusion with the legacy (pre 2009) render pass feature. In the present document however, the term *render pass* is always taken to mean the new render pass framework, which is supported by mental ray for Maya. The old render pass feature remains accessible through the render layer attribute editor's *Render Pass Options* section, but it is considered obsolete for mental ray since it is less powerful, less versatile, and slower. The old render pass feature is still relevant for Maya users who render with the Maya Software renderer.

2. Advantages of Using Render Passes

The compositing techniques that can be achieved using render passes were possible in the past, but may have been prohibitively complex and costly because of render layer management challenges and, in many cases, the need for user-written shader code. Render passes were primarily designed to help solve those problems and thus make advanced compositing workflows accessible and affordable to a larger class of Maya users.

2.1. Easier Workflow in Maya

The Passes tab in the Render Settings window allows users to create and configure render passes without any scripting or programming. Furthermore, the render pass Attribute Editor exposes a series of advanced options to tune the behavior of render pass extraction without having to get involved with shader code.

2.2. Interoperability

By using *Export for Precompositing* in the *Render* menu, it is more quick and easy to interchange and synchronize rendered frames between Maya and compositing applications. The Maya® Composite high-performance compositor (formerly known as Autodesk® Toxik™ software, and now a feature introduced in Autodesk Maya 2010 software) has built-in support for the *.precomp* interchange format. This file format is in fact a self-documented Python® script, making the data more easily interpretable by user-written scripts and third-party applications. Instructions on using this feature can be found in the Maya User Guide, under Rendering and Render Setup > Rendering > Rendering menus > Render > Export Pre-Compositing.

2.3. Lower Render Times

Because multiple render passes can be computed simultaneously, rendering multiple render passes is usually much faster than rendering the same number of render layers, which are rendered sequentially. Due to the new Maya shader architecture, it is not necessary to re-evaluate rays and shaders multiple times for all rendered images.

2.4. Reduced Necessity for Custom Shader Development

The Maya base shaders have native support for a wide range of built-in render passes, making it rarely necessary to edit and recompile shaders to extract information useful for compositing.

2.5. Faster and Easier Material Shader Authoring

Maya 2009 and later comes with a Shader SDK that helps make it more quick and easy to write new material shaders and light shaders that support the render pass framework. Furthermore, our C++ template-based architecture helps make it possible to re-use parts of pre-existing shaders without the need for code duplication or reverse-engineering.

3. Render Layers *vs*. Render Passes

In Maya, there are two similar but distinct means of producing decomposition images: passes and layers. Render layers are intended for decomposing scenes at the object level and for overriding properties, while render passes are intended to decompose data at the shading level. However, shading decompositions can be achieved with render layers by using the material override and attribute override mechanisms, and object level decompositions can be achieved in render passes by using pass contribution maps (discussed further in section 4.1). The selection of the method for achieving a given decomposition is situation dependent.

3.1. Reasons to Use Render Layers for Decomposition

3.1.1. Scene Partitioning for Performance

Very large and complex scenes often test the limits of what a graphics workstation or render node can handle, especially in terms of RAM capacity. Despite the additional address space of 64-bit architectures, physical RAM remains limited. A common solution is to break down the scene into layers that are rendered separately, then composited. The renderer only needs to process the scene entities present in the layer being rendered, which makes render layers a good tool for breaking down the computational burden. This is known as the *divide and conquer* strategy.

3.1.2. Overrides

The layer override mechanism is a powerful tool that allows node attribute values and connections to be different from one layer to another. There is no equivalent for render passes. However, there are many uses for layer overrides that can be replaced with alternate pass-based workflows. For example, material overrides with a black surface shader could be used to perform hold-outs. On the other hand, the same effect can be achieved by hiding the hold-out object using a pass contribution map, and turning-on the hold-out option in the pass Attribute Editor. The equivalent pass-based workflow is usually preferable because it is typically easier to set-up and it is usually faster to render multiple passes than multiple layers.

3.1.3. Pre and Post Render Scripts

The pre and post render scripts allow for a powerful level of render customization. They help make it possible, among other things, to temporarily alter a scene for a specific render layer, and restore the scene when the render is done. This is not possible with render passes because they are rendered in parallel.

3.1.4. Camera and Lens Effects

Many effects are achieved through the manipulation of eye rays[1], which is performed by the virtual camera or a lens shader. Because render passes that are rendered simultaneously share the same rays, it is not possible to vary eye ray based effects (*e.g.* motion blur, depth of field, lens distortion) on a per-pass basis. Many lens shaders also apply color transformations such as exposure control and tone mapping, which could, in theory, be controlled on a per-pass basis, but that feature is not available as of Maya 2011. Currently, Maya lens shaders that apply color transforms will only affect the Master Beauty pass. This is by design because **tone mapping should normally be applied downstream of compositing since render pass compositing should always be performed in a linear color representation** for reasons of simplicity and correctness.

3.1.5. Global Illumination and Final Gathering

Global illumination and final gathering computations are based on the set of lights and geometries present in the current render layer. Photon effects and final gathering may not vary on a per-pass basis because the associated computations are only performed on a per-layer and per-camera basis. Therefore, render layers can be used to help isolate the indirect illumination from a specific light or group of lights.

3.2. Reasons to Use Render Passes for Decomposition

3.2.1. Performance

Because concurrent render passes are computed simultaneously, a lot of ray-tracing and shading computation duplication can be avoided by using render passes. Render passes also help avoid the duplication of many preliminary computations such as tessellation, and the generation of global illumination maps, final gather maps and shadow maps. With a few exceptions, rendering multiple render passes within the same render layer is much faster than rendering the same number of single-pass render layers, as long as there is ample RAM to accommodate the additional buffers required by the render passes. Another performance benefit of render passes is that the scene needs to be translated once per render layer; so reducing the number of render layers by using render passes may decrease translation overhead. Translation is the process of converting the Maya scene into a data representation understandable by the renderer, which can be a significant computational burden in many cases.

[1] Eye ray: Straight line path from an imaginary observer's eye (or camera), through a screen pixel (or sub-pixel sample), into a virtual 3D world. Casting eye rays is a fundamental part of the ray-tracing rendering technique.

3.2.2. Scene Segmentation with Optical Interactions

In Maya, there is a node type called passContributionMap, which is used for per-pass scene partitioning. Scene entities that are excluded by a render layer are bypassed in the scene translation process, which means the renderer is unaware of their existence at render time. On the other hand, objects that are present in a render layer, but excluded from a render pass belonging to that layer, are available to the renderer, and thus may optionally be used in optical interactions even though the objects are not rendered. For instance, a render pass may be configured such that hidden objects continue to cast shadows, be visible in reflections and refractions, or produce hold-out silhouettes.

Light contributions from global illumination and final gathering cannot be controlled per pass because they are computed only once per render layer. Therefore, excluding an object at the pass level means that the object still produces indirect illumination effects (caustics, color bleeding) on surrounding objects. In order to suppress an object along with its indirect illumination effects, it must be excluded at the layer level.

3.2.3. Render Pass Types and Options

The main objective for using render passes is the functionality of the passes themselves. The various render pass types are programmed to extract specific decompositions of the computations made in the material shader, or other shading state information, in a highly configurable manner without any programming or complex shading network design. Built-in render pass types are enumerated and described in the Maya 2011 User Guide, under Rendering and Render Setup > Rendering > mental ray for Maya rendering > Visualize and render images > Multi-render passes. Some of the render pass configuration options are explained later in section 6, and they are formally documented in the User Guide, under Rendering and Render Setup > Rendering > Rendering nodes > Render pass nodes > Render pass Attribute Editor.

3.2.4. Sample Coherence

Producing many render passes in the same render layer assures that each of those passes will be constructed from the same set of samples (i.e. from the same eye rays, secondary rays, and light/shadow rays). On the other hand, the image sampling in separate render layers may differ because of adaptive sampling or inconsistent pseudo-random effects. This may result in sub-pixel inconsistencies, which may yield artifacts when compositing images from different render layers.

For example, if a scene element is submitted to different illumination conditions from one layer to another, this may affect the adaptive refinement of eye ray and/or shadow ray sampling, leading to subtle discrepancies between render layers. These discrepancies are not always noticeable at first, but they sometimes lead to compositing artifacts down the road.

This problem can be reduced by boosting sampling settings and/or not using adaptive sampling, at the cost of decreased rendering performance. Alternately, the problem can be avoided altogether with render passes.

3.3. Grouping Render Passes

Render passes and render layers are both represented by Maya nodes and links can be created between layers and passes. These associations are exposed in the Passes tab of the Render Settings window. For a render pass to be rendered, it needs to be associated with one or more renderable render layers. When a given render layer is rendered, all of its associated passes are rendered. Therefore, when a new render pass is created, it can be set-up to be rendered with any number of render layers in a scene.

3.3.1. Grouping by Layer

In many situations, it is desirable to separate render passes so as to not render them all in the same render layer. An obvious reason for that is when layer overrides or geometry exclusions are necessary to set-up the passes correctly. However, there are other

reasons for organizing a large set of passes into multiple layers, such as performance and workflow.

Performance wise, it is usually beneficial to group many render passes into a single layer to have them render simultaneously. However, each additional render pass may increase the amount of memory consumed by the renderer at render time. Therefore, having too many render passes in the same layer may result in a slow-down due to virtual memory swapping, and even out-of-memory errors, especially with 32-bit operating systems. An interesting solution to avoid running out of memory address space is to use the *frame buffer file* feature of mental ray (*c.f.* mental ray manual). Another trick is to decrease the number of render threads in order to reduce the mental ray render's RAM footprint. Using fewer threads is counter-intuitive, but it often helps increase performance when memory contention is the main performance bottleneck. Ultimately, one should use a divide and conquer strategy consisting of breaking down the rendering into several layers, with fewer passes per layer.

Also, when going through rendering-compositing iterations, it is not always necessary to re-render all of the passes. For instance, when adjusting light attributes, only the passes affected by the edited light need to be re-rendered. Therefore, grouping passes into layers based on their usage can help avoid wasting CPU-time on the render farm. The alternative is to use the *Renderable* attribute on render pass nodes.

3.3.2. Grouping by Set

Another means of grouping render passes is through a node type called RenderPassSet. Render pass sets are hubs for the links between layers and passes. They are intended to simplify pass management by allowing the user to assign render passes in groups. Sets can be organized in multi-level hierarchies.

4. Render Pass Principles

4.1. Understanding Pass Contribution Maps

The pass contribution map[2] is one of the most fundamental tools for managing render passes. Nodes of this type act as connection hubs between DAG objects and render passes. By default, the set of DAG objects belonging to the current render layer are rendered in a render pass. If one or more pass contribution maps (PCMs) are connected to a render pass, only the DAG objects of the current render layer that are connected to one or many of those PCMs are rendered. PCMs that are not connected to the current render layer are ignored.

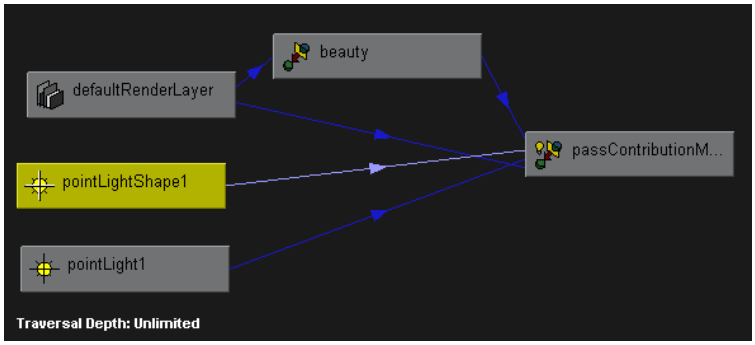


Figure 1 A Light Connected to a PCM

Two categories of objects are managed by PCMs: renderable geometries (e.g. surfaces, volumes) and light sources. Geometries are connected via their transform nodes, and instances with DAG paths that traverse a transform node connected to a PCM are considered members of that PCM. That makes it possible to connect individual instances as well as object groups. On the other hand, light sources are connected directly through

[2] A PCM, or pass contributions map, is a type of Maya scene node. Instructions on using PCMs are provided in the User Guide under: Rendering and Render Setup > Rendering > mental ray for Maya rendering > Visualize and render images > Multi-render passes

their shape nodes, which is consistent with the policy that light instancing is not officially supported in Maya. Some light sources, such as area lights, are also renderable geometries, and can therefore either be connected as a geometry (via a parent transform node) or as a light (via the shape node). When a light is initially added to a PCM, it is connected as both a light source and a geometry.

If the desired behavior is for an area or IBL light to contribute only as a light source or only as a geometry[3], then one of the two links must be deleted, which can be done in the Hypergraph editor or in a Maya Embedded Language (MEL) or Python script. In the case illustrated above, the light is a point light, so the connection with the transform node (pointLight1) has no visible effect on the rendered image.

The pass contribution assignment logic is inclusive, which means that a pass connected to multiple PCMs includes the union of all lights and geometries included by the PCMs. An additional subtlety is that for a PCM to be active, it must also be connected to the current render layer. This adds an additional level of control that allows the same render pass node to be used from one render layer to another, with different pass contribution maps, without relying on the layer override mechanism.

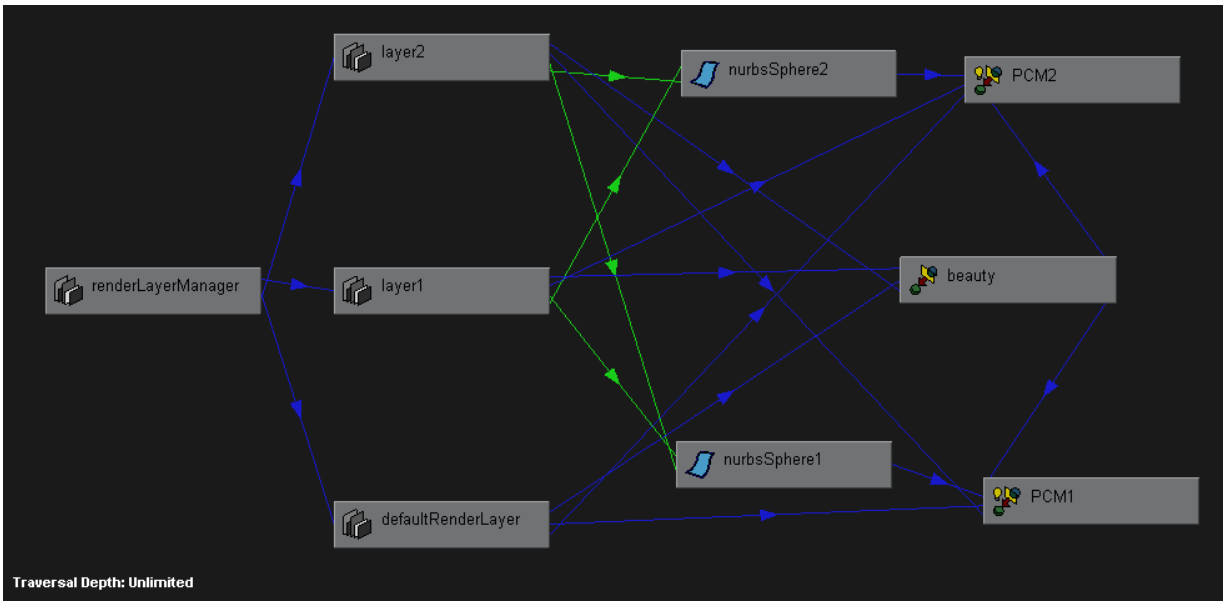


Figure 2 PCM variants per layer

In the example of figure two, there are two spheres that are members of all layers, and connected to their respective pass contribution maps. There is also a beauty pass that is associated with all layers and both pass contribution maps. However, the beauty pass contains different geometry based on the render layer: the instance of the beauty pass for defaultRenderLayer contains both spheres because both PCMs are connected to defaultRenderLayer; the beauty pass for layer1 contains only nurbsSphere2; and the beauty pass for layer 2 contains only nurbsSphere1. These associations are controlled through two user interfaces (UIs) in Maya: the Render Layer Editor, and the Passes tab of the Render Settings window. See the Maya User Guide for more details on how these interfaces are operated.

Another subtle behavior about PCMs is that if none of the PCMs attached to a given pass contain any lights, then all lights in the current layer are turned-on. This simply means that a user who only wants to use PCMs to control geometry contributions does not need to bother with attaching lights to PCMs. The same is also true for geometries, hence if

[3] The 'geometry' aspect of rendering a light source refers to the surface representing the light source being rendered as an emissive (i.e. incandescent) object. This only applies to area lights and IBL.

none of the PCMs attached to a given pass contain any geometry, then all geometry in the current render layer is turned on. To understand this logic, it is important to remember that by default, lights get connected as both light sources and geometries. Therefore, the links to the light transform nodes need to be removed in order for PCMs to only control illumination contributions. One way to remove a light's transform-to-PCM connection by hand is to:

1. select the light in the viewport,
2. display the *Hypergraph Connections* window,
3. select and delete the transform-to-PCM connection.

For example, the transform-to-PCM connection is the one selected (highlighted in yellow) in Figure 3.

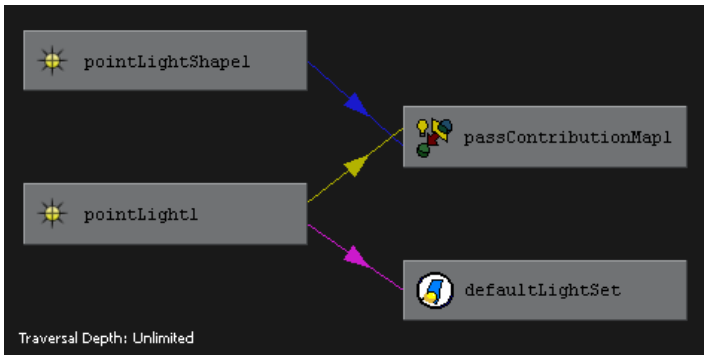


Figure 3 Removing the Transform-to-PCM Connection of a Light

PCMs and light linking can be used together. For scene geometry to receive illumination or cast a shadow from a given light source in a given render pass, both the PCM conditions (specific to the render pass) and the light linking conditions (specific to the render layer) must be met.

An important limitation of PCMs is that they are based on DAG object connections rather than objectSets. Maya object sets are capable of component level ownership (individual faces, edges, vertices), while PCMs are not. Therefore, an individual DAG shape instance is either completely or not at all a member of a PCM.

4.2. Dealing with Shadows

Pass contribution maps may be used to control not only which objects are visible, but also which objects cast shadows. Depending on how a render pass is setup (more on pass options in section 6), objects that are present in the current render layer, but not in the current render pass, may cast shadows onto objects that are included by the render pass. This can be very useful for many compositing workflows. However, it should be noted that this level of control over shadows is only provided for ray-traced shadows.

Shadow maps provide a means to accelerate rendering, but they have the limitation that they are rendered only once per camera per render layer. Therefore, they cannot capture per-pass shadow casting differences. The same is also true for global illumination and final gathering, which may produce indirect shadows.

4.3. The Master Beauty

The Master Beauty is the default pass, and it cannot be controlled explicitly. It helps provide backwards compatibility with Autodesk® Maya® 2008 software. There is no node for it, so it cannot be deleted or configured, and it cannot be attached to a pass contribution map. In the mental ray embodiment of the render pass system, this pass uses the main *RGBA* buffer, while all other render passes use custom frame buffers. The main *RGBA* buffer is the only buffer in mental ray that cannot be turned off.

5. mental ray for Maya Render Pass Fundamentals

5.1. Custom Frame Buffers

By default, mental ray rendering software produces a single result image per render. In the mental ray documentation, this built-in buffer is referred to as the main *RGBA* buffer. In Maya (since version 2009), this buffer is known as the Master Beauty pass. mental ray allows the user to define additional auxiliary buffers, i.e. custom frame buffers, in order to store additional results generated during rendering. Writing data to these additional buffers requires shaders that were purposely designed to do so.

mental ray for Maya uses the custom frame buffer feature of mental ray to render an arbitrary number of passes simultaneously. In previous versions of Maya (prior to 2009), it was possible to set-up custom frame buffers to allow user-written or third-party shaders to produce extra images. In order to prevent conflicts with pre-existing shaders, it is still possible to manually reserve custom frame buffers for this purpose. This is exposed in the Attribute Editor of the miDefaultOptions node, under the Extra Attributes tab. That workflow is deprecated and is maintained solely for backwards compatibility. The preferred method of implementing custom passes, as of Maya 2009, is to append new outputs to the output structures of custom shaders. Those new outputs can in turn be connected to a *writeToColorBuffer* shader. The writeToColorBuffer shader can then be associated with a pass of the CustomColor type. This method provides the user with geometry-wise pass contribution map support for free. Several behaviors of the writeToColorBuffer shader were improved in Autodesk® Maya® 2010 software, and performance was improved in Maya 2011. A patch for Maya 2010 with the performance improvement is available to subscription customers upon request.

5.2. Pass Implementation Categories

In the mental ray for Maya plug-in, not all render passes are implemented the same way. It is important to understand the subtleties of how passes are implemented in order to better comprehend how they work.

5.2.1. Light Loop Material Passes

This category of render passes is handled by material shaders inside their respective light sampling loops. Built-in render pass types that depend on direct illumination are implemented this way. The render pass types in this category are:

- Ambient (point-light-like part of ambient light contribution)[4]
- Ambient Irradiance
- Beauty (direct illumination portion)
- Beauty No Shadow (direct illumination portion)
- Diffuse
- Diffuse No Shadow
- Direct Irradiance
- Direct Irradiance No Shadow
- Raw Shadow
- Shadow
- Specular
- Specular No Shadow

[4] The Maya ambient light behaves as a combination of a point light (shading contribution computed in the shader's light loop) and an omni-directional light (computed outside of the light loop).

- Translucence
- Translucence No Shadow

Because the passes are evaluated in the light sampling loop, each pass can have a different illumination value per sample, allowing pass contribution maps to be taken into account. For instance, a given light sample may be forced to zero for a given pass if the sample evaluates a light that is not included in pass contribution maps linked to that pass. At the same time, other render passes may be receiving light for the same sample. The same logic also applies for shadow casters, which can be included or excluded on a per-pass basis using PCMs.

5.2.2. Non-Light Loop Material Passes

Some render pass types are not light-dependent, so they are not in the light loop, but they are still dependent on data or intermediate results that can only be captured by the material shader. These pass types are:

- Ambient (omnidirectional portion of ambient light contribution)
- Ambient Material Color
- Beauty (indirect light contributions from GI, FG, scattering, incandescence, ambient)
- Diffuse Material Color
- Incandescence
- Incidence (Light / Normal)
- Indirect Irradiance
- Glow Source
- Material Normal
- Reflected Material Color
- Reflection
- Refraction
- Refraction Material Color
- Scatter

As a result of not being written from the light loop, none of these pass types are affected by PCM-based light inclusion/exclusion. Therefore, the scattering, final gathering, and global illumination contributions take into account lights that belong to the current render layer, regardless of PCM associations. The main reason for this design is that the computations involved are too expensive both in terms of CPU-time and memory to be performed on a per-pass basis.

5.2.3. Non-Material Passes

Non-material passes are passes that are not written by regular material shaders. Instead, they are written by special-purpose shaders that compute the value to be written to the pass's frame buffer directly from the ray state, without dependency on the material shading network. Although these pass shaders do not render materials, they are technically material shaders that are rendered sequentially with the shading engine shader. The shading engine is a special shader that is at the root of surface material shading networks, and is responsible for several low-level rendering tasks, such as compositing render pass contributions from child rays (reflections, refractions, and transparencies). Because the shaders responsible for non-material passes are independent of the shading engine, they will not be composited the same way material passes would be. Non-material pass types are:

- 2D Motion Vector
- 3D Motion Vector
- Ambient Occlusion
- Camera Depth
- Coverage
- Normalized 2D Motion Vector

- Object Incidence (Camera / Normal)
- Object Normal

Some of the advanced compositing options that are available with certain other render pass types are handled by the shading engine, and are therefore not available for non-material passes.

The fact that the render pass implementations are in separate shaders has important implications for the mental ray scene description. It means that the pass shaders need to be inserted into affected material blocks. For example, a material block with a camera depth pass would look like this in an .mi file:

```
material "phong1SG"
      "adskMayaShadingEngine" (
            "surfaceShader" = "phong1.outColor",
            "cutAwayOpacity" 0.,
            "customShader" off
      )
      "adskPassCameraDepth" (
            "frameBufferNumber" 0,
            "encodingIndex" 0,
            "remap" off,
            "znear" 0.,
            "zfar" 1000.,
            "minbuffer" 0.,
            "maxbuffer" 1.
      )
      shadow = "phong1:shadow"
end material
```

The shader calls to adskMayaShadingEngine and adskPassCameraDepth are untyped, which means they are considered material shaders by default. mental ray puts both calls into a linked list, and both shaders are called sequentially. This logic works well when rendering monolithic scenes. However, getting non-material passes to work using mental ray assemblies can be tricky because the frame buffers for the render passes are declared in the camera block, which is usually in the main scene file, while some material blocks may come from assemblies, which may have been exported with a different render pass configuration. As of Maya 2011, there is still no fool-proof solution to this problem. One approach would be to write a script to insert the necessary shader calls to material blocks in assembly files. To do so, it is necessary to understand many of the details of how Maya render passes are described in .mi files, which are explained later in section 8.

In a normal setup, all secondary rays are cast from the shading engine or one or many of its child shaders, which implies that the pass value written by the shallowest surface hit (in terms of ray recursion) prevails for pass shaders that are called after the shading engine.

5.2.4. Shading Engine Passes

Pass types managed directly by the shading engine are:

- Matte
- Opacity

These pass types are subject to being composited at render time, so they need to be handled under the shading engine, but their computation is independent of the material, so they do not need to be handled by individual material shaders. Their behavior is essentially identical to non-light loop material passes.

5.2.5. Volume Passes

The volume engine[5] takes care of contributing volume rendering results to various render passes. The pass types that receive contributions from the volume engine are:

- Beauty
- Light Volume
- Object Volume
- Scene Volume

Volume rendering effects contribute to beauty passes. The other volume passes help isolate specific types of volumetric effects. The Light Volume pass type captures light fog effects; Scene Volume is for global fog (*e.g.* atmospheric effects); and Object Volume is for other volume rendering effects (volume fur, fluids, particles, *etc*.)

5.3. Post-Processing Effects

With mental ray for Maya, output shaders are used to help produce various post-processing effects. These effects are not captured by render passes. Maya render passes were designed to help solve compositing interoperability workflow issues. In a production pipeline that involves compositing rendered images, it is not usually appropriate to apply post-processing effects in the rendering stage because then the effects become baked-in.

5.3.1. Glow

Glow is a good example of a basic post-processing effect that is built-in to Maya. This effect is provided for convenience to allow users to produce a lens glow effect directly at render time. Users that use compositing in their production pipeline should not use this feature because pre-baked glow is notoriously hard to composite correctly. Before discussing glow workflows any further, the following provides some theory about how glow works.

5.3.1.1. Real-World Glow

Glowing is a real-world optical phenomenon that causes cameras and human observers alike to see light halos around bright objects. The two most common phenomena that cause glow are: light scattering in a volume surrounding the light source (like a bolt of lightning in a cloud), and lens flares. The first type of glow is a rendering effect that can be achieved through volume rendering. On the other hand, lens flares are the type of glow that are commonly rendered as a post-processing effect because they can be accurately simulated on a projected 2D image without any knowledge of the composition of the 3D scene.

Lens flares occur because the light entering a lens does not all follow the intended refractive path. Some portion of the light may be reflected, diffracted and/or scattered. The proportion of light that follows unwanted image formation paths is usually so small that the effect is not visible in photographs of low dynamic range (LDR) scenes. The artifacts occur when a particular object in the scene is much brighter than the rest, such as the Sun.

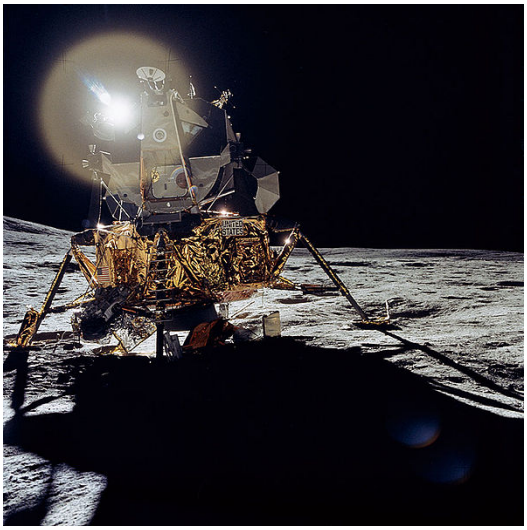


Figure 4 Example of Lens Flare in a Photograph[6]

[5] The volume engine is a built-in shader. It is used by mental ray for Maya for managing volume rendering effects. Maya base volume shaders are designed to be used in conjunction with the volume engine.

[6] Source: NASA (public domain)

In the above photograph, the lens flare appears overlaid on top of the foreground objects despite the fact that the sun, which is causing the flare, is further away. This is because the physical phenomenon causing the flare occurs in the image formation process, and not in the scene.

5.3.1.2. Rendered Glow

The glow pattern around the sun in Figure 4 is caused mostly by scattering, and can easily be modeled as a stationary impulse response. This is the type of artifact that is modeled by the camera glow effect in mental ray for Maya. More complex flare patterns caused by light reflection (typically circles and rings lined-up along an axis) are more difficult to model because the response varies with the position of the light source.

The reason for computing glow as a post processing effect is that the light from a simple image sample (eye ray) can affect a large area of the final image, which makes it impractical to compute the effect during the ray-tracing stage of rendering. It makes more sense to compute the effect by filtering the rendered image.

In high-dynamic range (HDR) images, a glow effect that accurately models a “scattering” flare is very simple to produce using a regular discrete convolution filter. However, the camera glow effect in mental ray for Maya is a little more involved than that because it was designed to be able to work with low dynamic range image buffers. Because of the color clamping that occurs when rendering to an LDR image buffer, it is impossible to determine *a posteriori* the real brightness of an object that is rendered as full white. Therefore, it is not possible to accurately determine the intensity of the glow pattern that would appear within the dynamic range of the image. To overcome this problem, mental ray for Maya uses a glow buffer to store the intensity of the glow produced by each image sample. This is a value typically within the dynamic range of the image buffer, and is stored in a separate image buffer. mental ray for Maya implicitly creates a single glow buffer that is used to accumulate glow values that are used to modify the Master Beauty pass as a post processing effect. The glow post process consists of simply applying to the glow image a convolution filter corresponding to the glow pattern, and adding the result to the Master Beauty buffer.

5.3.1.3. Producing Near Realistic Glow Effects

With Maya base shaders, glow is controlled by the “glow intensity” attribute on material shaders. This is a scale factor that is used to multiply the shader’s output color, to produce the value to be stored in the glow buffer. The glow intensity value should typically be near zero, to bring the high intensity glow-producing regions of the image into the glow buffer’s dynamic range. In theory, to mimic the behavior of a real lens, the same glow intensity value should be systematically applied to all materials in the scene. The main reason this parameter is a material shader property rather than a camera property in Maya is to provide artistic freedom to the user, rather than to strictly enforce physical correctness.

5.3.1.4. The Glow Source Render Pass

The output shader that is currently used to process the glow buffer does not interact with the Maya render pass framework. It only modifies the Master Beauty pass to produce a baked-in glow effect that is not convenient for downstream compositing. In order to add in a glow at the compositing stage, there are several possible workflows. When working with HDR images, most compositing software applications have tools to directly compute glow or other types of lens flare effects without requiring any additional input. With LDR images, however, one can reproduce the same technique implemented in the Maya software’s built-in glow by using a Glow Source render pass. Render passes of this type are equivalent to the glow buffer described above, with the added flexibility of PCM support. In order to produce a glow effect from a glow source pass, the image must be filtered using a filter representative of the desired glow pattern (a blurring filter for example), before it can be composited additively. This workflow allows the artist to control

the glow pattern and intensity interactively in the compositing software, without ever having to re-render the scene.

Alternatively, it is possible to isolate the glow effect as produced by the Maya software's built-in camera glow feature in order to apply it in compositing. However, this cannot currently be achieved by using a render pass. Instead, the user must create a separate render layer, and use a layer override to turn on the *Hide Source* option on all materials. Then, the Master Beauty pass of that render layer will contain only the glow effect.

5.4. Complex Shading Networks

All material passes (light loop and non-light loop) are written to by material shaders. This can cause conflicts in complex shading networks that contain multiple material shaders or utility shaders that modify the result of a material shader without modifying render pass contents. The following subsections illustrate a few challenging cases that require special attention.

5.4.1. Combinations of Multiple Shaders

The shading network in the following illustration includes four surface shaders, each of which writes their respective results to render pass frame buffers. The condition1 node switches between the Anisotropic shader and the Phong shader depending on whether or not normals are inverted, which produces a two-sided material. The result of that conditional is then blended with a red Lambert material, and finally a surfaceShader node is required as an interface between the blend node and the shading engine.

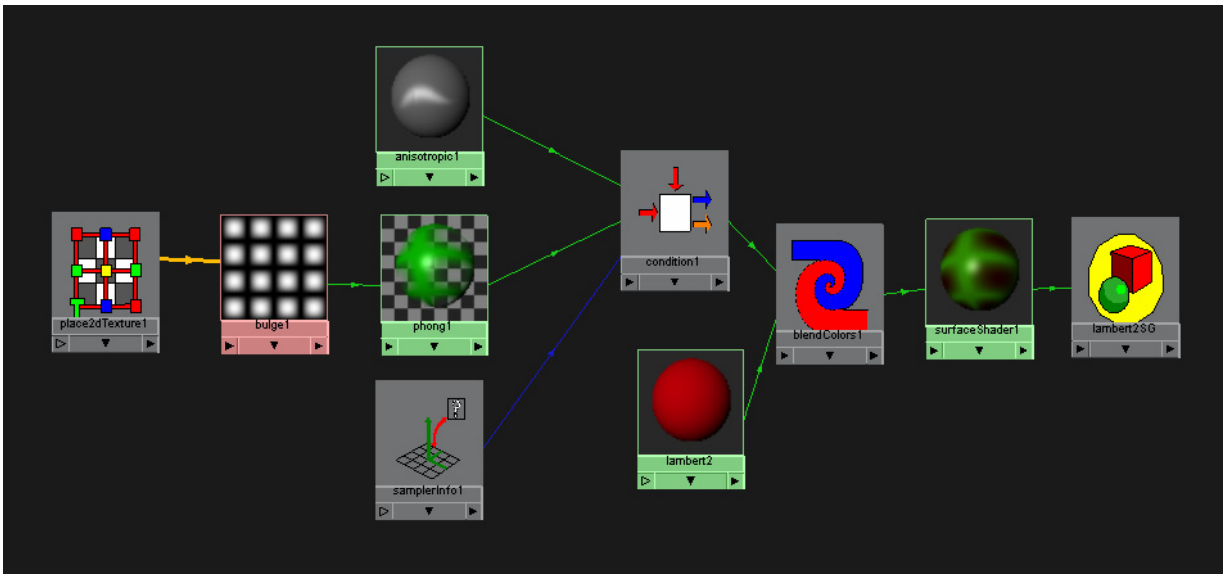


Figure 5 Shading network with blending and conditional

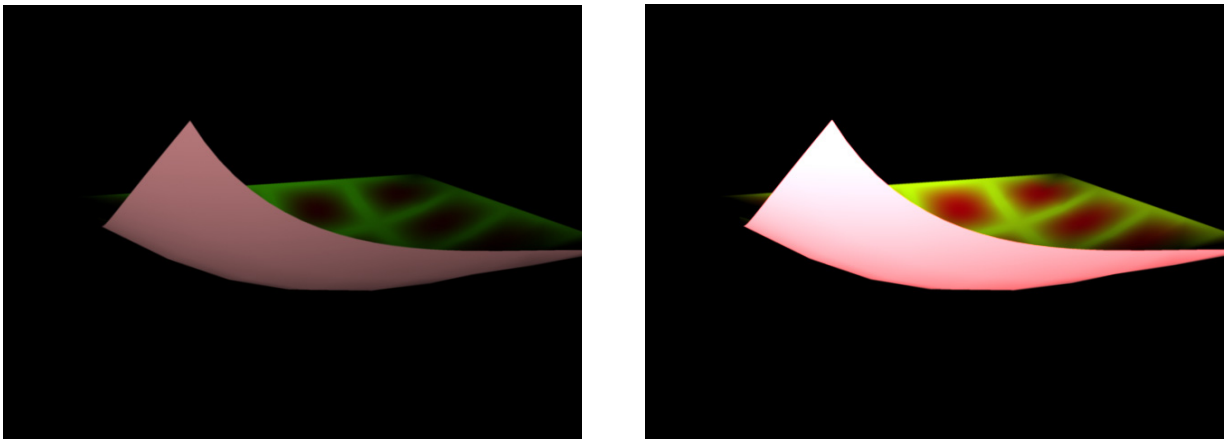


Figure 6 Rendering Results. Left) Master Beauty Right) Beauty Pass

As demonstrated in Figure 6, the Master Beauty renders as one would expect, but not the beauty pass, and this is because multiple material shaders are concurrently writing to the render pass frame buffers. To workaround this, edit the framebuffer write options on each of the shaders so that the sum of the values contributes to the frame buffer and respects the logic of the shading network. Naturally, this problem only happens with material passes (light loop and non-light loop). Let us examine three different ways to approach the problem.

Solution 1: Let only the surfaceShader1 node contribute. To do this, the frame buffer contributions made by the three other shaders must be turned off by setting the frame buffer write operation to "No Operation" in each of the shaders' attribute editors. This is a simple solution, but it only works for beauty passes, because beauty is the only render pass type written to by the surfaceShader node type, since it is just a pass-through shader. If the goal is to extract diffuse, specular, irradiance, and other material passes, this solution does not suffice. Another limitation of this method is that pass contribution maps do not affect lights and shadows because the active light loops are in the upstream shaders, not in the surfaceShader1 node, from which the contribution is taken. Despite these drawbacks, this method is still useful for a variety of important applications, such as geometry-wise partitioning and hold-outs using PCMs, as well as for isolating specific reflection and refraction trace levels (discussed in section 6).

Solution 2: The frame buffer contributions made by the surfaceShader1 node are turned off, and frame buffer contributions of the other surface shaders are scaled to match the effect of the blendColors1 node. In Maya 2011, the scaling is already done automatically by default (with the *Render Pass Mode* set to *Apply to Render Passes* on the BlendColors shader node). To do the same in Maya 2009 or Maya 2010, the frame buffer contribution scales of the upstream shaders need to be set manually, and frame buffer contribution scaling must be turned on.

With Maya 2009 or Maya 2010: In the above Hypershade graph, the lambert2 node is connected to the first input of the blendColors1 shader, so it has to be scaled by a factor equal the blender parameter of the blendColors1 shader. On the other hand, the phong1 and the anisotropic1 shaders need to be scaled by 1-blender. There is no need to worry about the Phong and Anisotropic shaders conflicting in this example because the condition node only evaluates the input that is selected by the condition, and the other shader does not even get executed.

With this solution, material render passes produce correct results.

Solution 3 (for Maya 2009 and Maya 2010): Link the blender parameter to the scale factors. This is equivalent to solution 2 except that the scale factors are kept in synch with the *blender* parameter through connections, which makes it convenient to adjust, animate, or texture-map the blender parameter. A node of type plusMinusAverage can be used to

compute the 1-blender factor needed for the shader that feeds into the second input of the blendColors1 shader. The plusMinusAverage node is set to have to 1D inputs and is set to subtract mode. Input 0 is set to 1, and input 1 is connected to the *blender* parameter of the blendColors node.

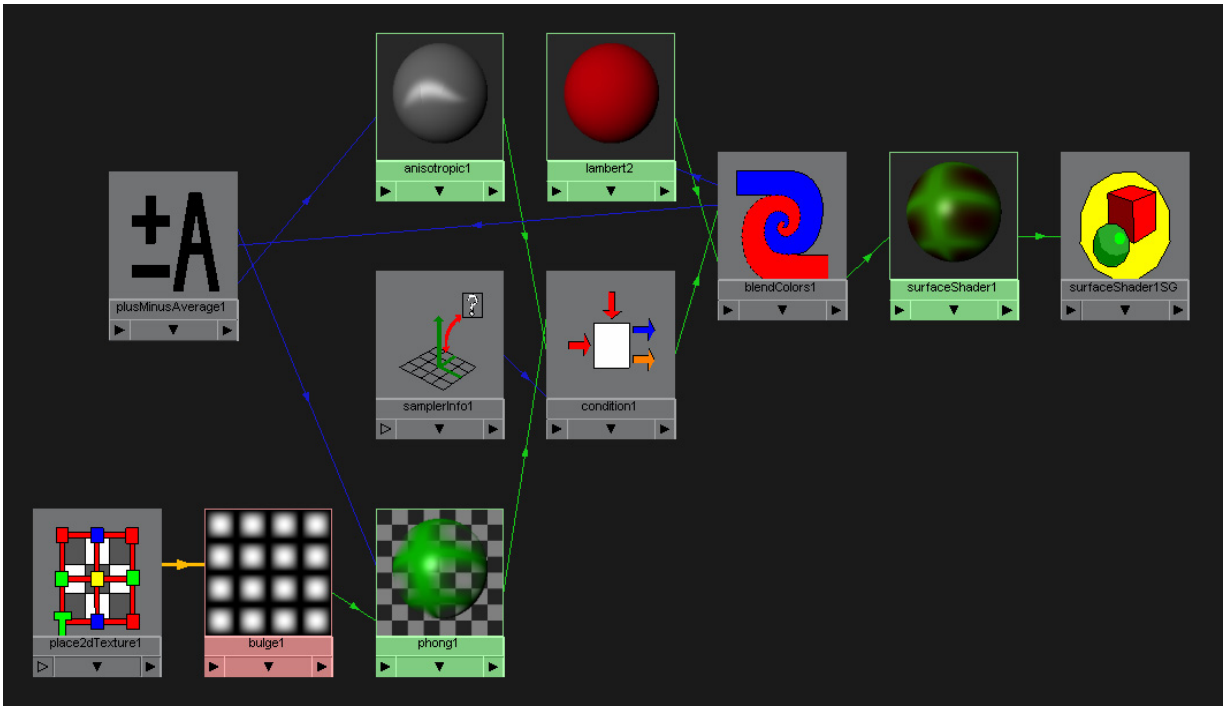


Figure 7 Shading network for Solution 3

New in Maya 2011: This class of problem has a much simpler solution as of Maya 2011, because the blendColors shader now has a Render Pass Mode parameter. All standard Maya shaders that are used for combining or manipulating colors also have this parameter (i.e. blendColors, layeredShader, remapColor, remapHsv, clamp, contrast, gammaCorrect, hsvToRgb, rgbToHsv, luminance). The available modes are:

- Pass through: This is equivalent to the Maya 2009/Maya 2010 behavior. The shader does not affect render passes. Shader operations performed by upstream shaders are cumulative.
- Apply to Render Passes: This is the default. Whatever color manipulation that is performed by this node (for the Master Beauty) is also performed on material render passes.
- No Contribution: Render pass contributions performed by upstream shaders are discarded.
- Write Shader Result to Beauty Passes: Beauty passes will receive a contribution corresponding to the shader's output value, which is normally the value used for computing the Master Beauty pass.

5.4.2. Chained Material Shaders

In some cases, it may be desirable to feed the result of one surface shader into another in order to achieve a creative effect. The Ramp Shader, for instance, could be used to generate values for the color parameter of a second shader in order to achieve a customized anisotropic coloring effect. This yields the exact same problem as the previous example because there are several surface shaders in the same network, all making independent frame buffer contributions. In most cases, the solution is as trivial as turning off frame buffer contributions on surface shaders, except for the shader at the root

of the network, which is typically the shader responsible for the actual shading (in the physical sense of the word) computation.

However, it may sometimes be useful to isolate render pass contributions from several different surface shaders in the shading network. Unfortunately, this cannot be done on a per-pass basis. The solution is to create multiple render layers with layer overrides on the frame buffer write operation attributes to only accept contributions from one of the shaders at a time. An alternative is to capture intermediate values from the shading network using the writeToColorBuffer shader for producing custom passes.

5.4.3. Non-Linear Color Transformations

The pass contribution scale factor, discussed previously, is very useful for reproducing the effect of a linear color operation (*e.g.* a blend), onto render passes. Unfortunately, a scale factor method shown in 5.4.1 is useless for handling non-linear color transforms, as in the shading network illustrated below.

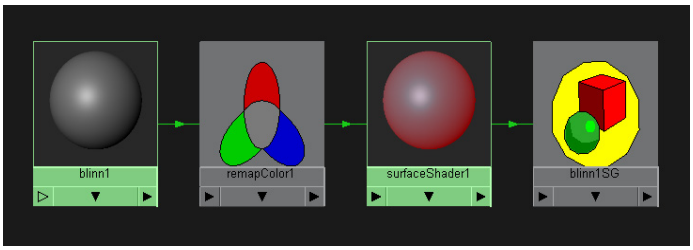


Figure 8 Simple shading network with a non-linear color transform

In this case, frame buffer contributions for one of the two surface shaders must be turned off in order to avoid a contribution conflict. By isolating the contributions from the blinn1 shader, passes can be generated without the effect of the remapColor1 node. In Maya 2011, the *Render Pass Mode* of remapColor1 must be set to *Passthrough*.

On the other hand, isolating the contributions from the surfaceShader1 node would not allow the capture of material passes except for beauty. This may seem like a serious problem, but it is not really, since extracting other material passes with the effect of remapColor1 is virtually useless from a compositing correctness stand-point. This is because non-linear color transforms are not distributive with respect to the decompositions provided by the other render passes. In other words:

$$T(a+b) \neq T(a)+T(b)$$

where *a* and *b* are colors, and *T* is a non-linear color transformation function.

Therefore, there is no easy and practical way to generate a beauty shot equivalent to the Master Beauty by compositing component passes that have been individually remapped. The proper workflow for production pipelines that require a decomposition into material passes, as well as color transforms, is to avoid performing the color transforms in rendering and to defer them to the compositing stage of the pipeline.

In situations where different color transforms need to be applied to different scene entities, render passes need to be segmented into one group per color transform, and compositing needs to be performed in two stages. In the first stage, the beauty for each group is composited from elementary material passes. Then, the resulting intermediate images are subjected to separate color transforms. Finally the color-transformed images are composited together to obtain the desired shot.

5.5. User-Written and Third-Party Shaders

mental ray material shaders that were not written using the API (application programming interface) provided by the AdskShaderSDK library do not write anything to render pass frame buffers. The long-term solution is to port those shaders to the SDK, or have them ported in the case of third-party shaders. However, even if a shader does not comply with the Maya render pass framework, it does not necessarily mean that there is nothing to be done with render passes. First of all, all non-material render passes work just fine. Also, there are several ways to pipe the output of non-compliant shaders into material render passes. However, because there is no way to intercept intermediate computations that happen inside a shader's light loop, it is not be possible to control light-ray-level

decompositions. That means that controlling lights and shadow caster contributions on a per-pass basis using PCMs does not work.

5.5.1. Passing Through a surfaceShader Node

A simple way to get a non-compliant shader to contribute to beauty passes is to hook it up to a surfaceShader node. In most cases, that is already necessary just to make the shader interface with the shading engine (i.e. the shading group node). This method is easier to set-up and may help satisfy many compositing use cases. For hold-outs to work correctly with semi-transparent geometry, it is important to remember to hook-up the transparency of the surfaceShader node, assuming that the non-compliant shader can be configured not to cast its own transparency rays, and let the Maya shading engine do the calculations. This can usually be achieved by specifying the transparency directly on the surfaceShader node and not on the 3rd party shader. An example of this workflow is given in section 10.4.

5.5.2. Capturing the Shader Output Structure

In many cases, non-compliant shaders may already be offering a decomposition of the shading computation in the form of an output structure. It is possible to re-direct these outputs into material render pass frame buffers, managed by the Maya render pass framework by connecting them to a special utility shader designed for this purpose. Such shaders are already in use in Maya for wrapping non-Maya-specific material shaders provided by mental images. The shader declaration file AdskShaderSDKWrappers.mi shows examples of phenomena that embody this strategy. Users may write their own adapter shaders using the shader SDK, or re-purpose the ones that are already provided with Maya. The provided adapter shaders were designed to serve as companions to specific material shaders, but nothing prevents them from being used for other purposes.

5.5.3. Custom Passes

Custom passes may be used as an alternative to using adapter shaders. The workflow is simpler, but there are important limitations. Custom passes are currently only evaluated on eye rays, and don't provide all the compositing options available with other pass types. On the other hand, custom passes require little effort to set-up.

Creating a custom pass is the same as creating any other pass type using the Passes tab in the Render Settings dialog. Depending on the type of data to be stored in the image, one of the four variants must be chosen (Custom Color, Custom Depth, Custom Label, or Custom Vector). Writing data into these buffers is achieved by placing writeTo*Buffer nodes in the shading network. For example, to write to a buffer of type Custom Color, one would use the writeToColorBuffer shader. In the shader's Attribute Editor, there is a field to select the corresponding type of buffer to be written to, which allows multiple custom buffers of the same type to co-exist. The resulting shading network would look something like Figure 9:

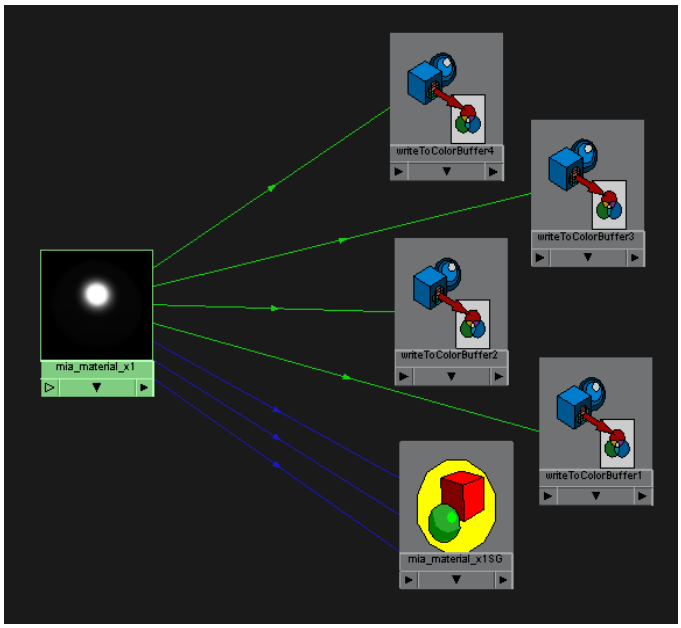


Figure 9 Shading Network for Writing to Custom Passes

It is important to note that the writeToColorBuffer nodes are not on the evaluation path that leads to the shading group node. Therefore, they are not evaluated under normal circumstances. However, if the Evaluation Mode parameter of

each of those shader nodes is set to *Always*, they are forced to evaluate anyways.

Using the *Always* mode is convenient, but it may result in significant performance degradation in Maya 2009 and Maya 2010. In the example of Figure 9, the mia_material_x1 material is evaluated twice for each hit: once for the regular shading network path, and once more because it is upstream of nodes that are in *Always* mode. This double evaluation is caused by a shader cache management issue, which is resolved in Maya 2011. With Maya 2009 and Maya 2010, the performance issue can still be resolved: the writeToColorBuffer nodes need to be chained using their evaluationPassThrough inputs and outEvaluationPassThrough outputs. They must be chained in a way that places them on the shading network evaluation path.

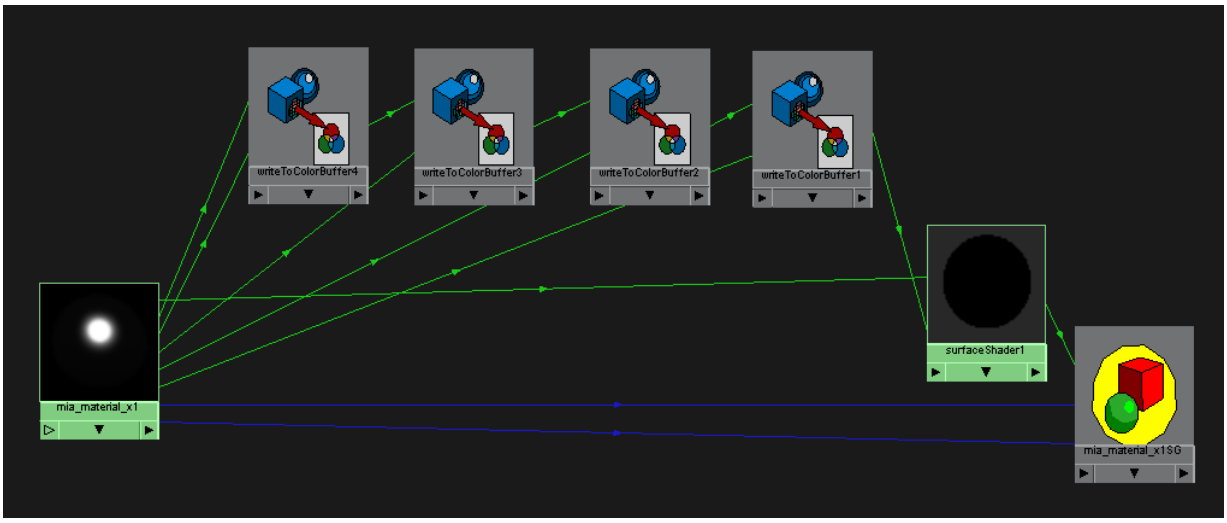


Figure 10 Optimized Shading Network for Writing to Custom Passes

In the example of Figure 10, the *result* output of mia_material_x1 is relayed to the surfaceShader1 node via the evaluationPassThrough connections, which triggers the evaluation of the writeToColorBuffer nodes. The material transparency is connected directly. The link to the miMaterialShader input of the shading group was broken to allow the surfaceShader1 node to be connected, but the connections to miShadowShader and miPhotonShader remain untouched. Also, the Evaluation Mode must be set to *Pass Through Only*. This workaround is obsolete in Maya 2011, but it is still supported for backwards compatibility.

5.6. Bypassing the Shading Engine

The shading group node has an option to *Export with Shading Engine*. This option is turned on by default. This option should only be turned off when using a custom or third party shader library that is not designed to work with Maya. The shading engine is required for material render pass contributions to be computed correctly. Without the shading engine, only the Master Beauty pass, custom passes, and non-material passes can be expected to render correctly. Also, writeTo*Buffer shaders with an evaluation mode set to *Always* will not be evaluated unless the shading engine is present.

6. Options for Material Passes

In the Render Pass Parameters section of the render pass node Attribute Editor, there is a series of contextual options. The options available depend on the render pass type. This section of the document looks at options commonly available for most types of material render passes. In order to illustrate how all of these options work, the following is a simple test scene that contains many optical effects with a single beauty pass.

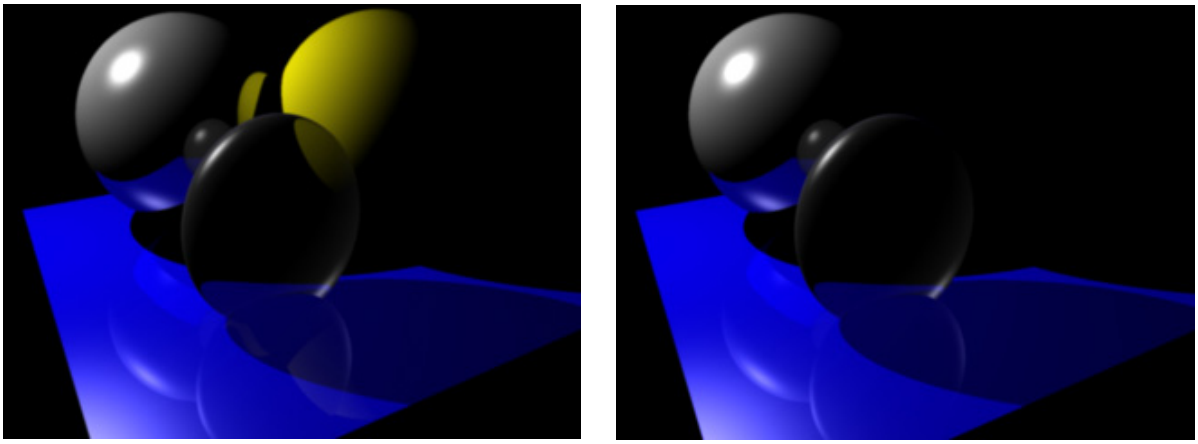


Figure 11 Test scene. Left) Master Beauty; Right) Beauty Pass

In the example scene, a pass contribution map was used to exclude the yellow sphere from the beauty pass. The render pass parameters for the beauty pass were left to their default values. These parameters may not have the desired effect when using shaders that are not fully compliant with the Maya render pass framework.

6.1. Shadows

This option indicates whether or not ray-traced shadows are computed for the current render pass. This is on by default.

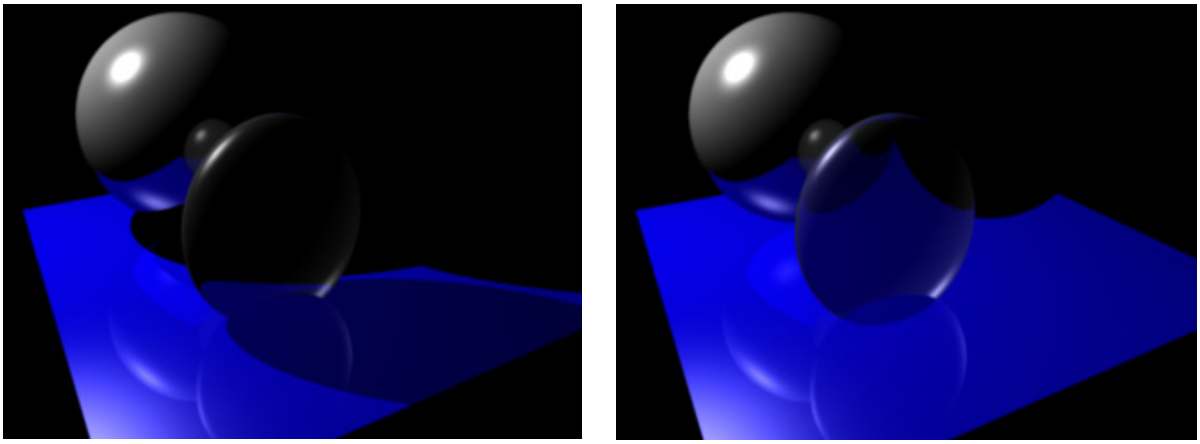


Figure 12 Shadows On vs. Off

6.2. Hidden Geometries Cast Shadows

This option indicates whether or not objects excluded by pass contribution maps but included in the current layer shall cast shadows. By default, this option is on, which explains why we see the shadow of the yellow sphere even though the yellow sphere is invisible.

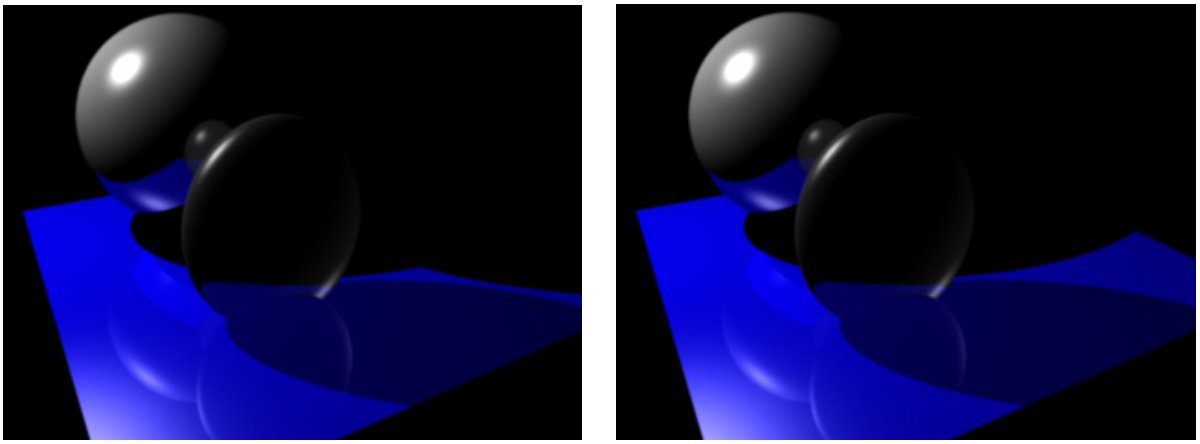


Figure 13 Hidden Geometries Cast Shadows On vs. Off

6.3. Hold-out

The hold-out option indicates whether or not geometries excluded by pass contribution maps occlude the scene. By default this option is turned on, which causes hidden objects to produce silhouettes. This option is very useful for compositing a partitioned scene, by simply adding the images.

When hold-out is disabled, hidden objects become invisible. In that case, compositing a segmented scene requires mattes and possibly depth buffers. The advantage of compositing with hold-out disabled is that occlusions are not baked-in, which allows the segments to receive independent geometric transforms (*e.g.* zooming and panning) and still produce a good composite. Also, turning hold-out off may slightly increase render time because some additional surfaces, that would otherwise be occluded, need to be rendered.

6.4. Use Transparency

This option indicates whether or not the scene is rendered with transparency. When turned off, all geometries are rendered as opaque. With mental ray for Maya, Refractions are not considered as transparencies. Therefore, refractive objects are not considered transparent, and continue to be see-through when this option is turned off.

6.5. Hidden Geometries Visible in Reflections

This option indicates whether geometries excluded from the pass by pass contribution maps, but present in the render layer, are visible indirectly through reflections.

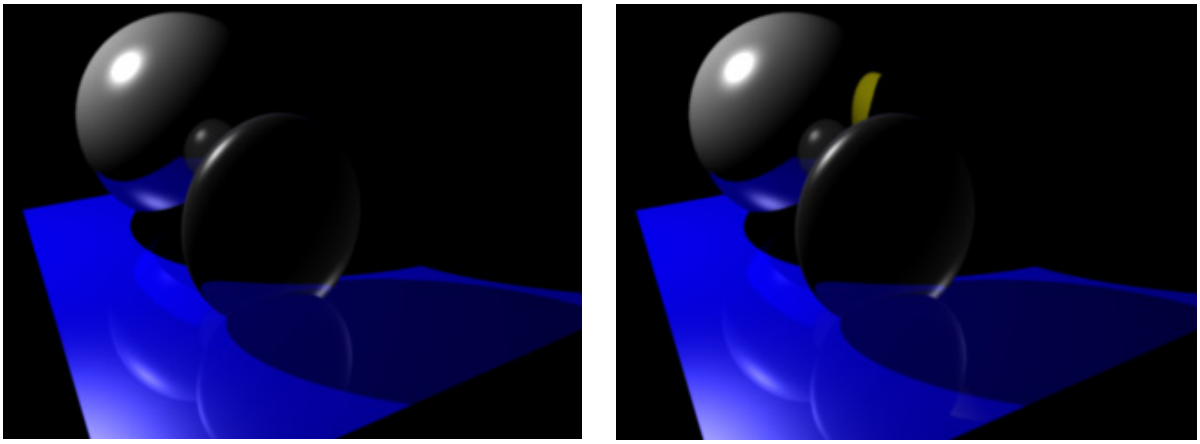


Figure 14 Hidden Geometries Visible in Reflections Off vs. On

6.6. Hidden Geometries Visible in Refractions

This option indicates whether geometries excluded from the pass by pass contribution maps, but present in the render layer, are visible indirectly through refractions.

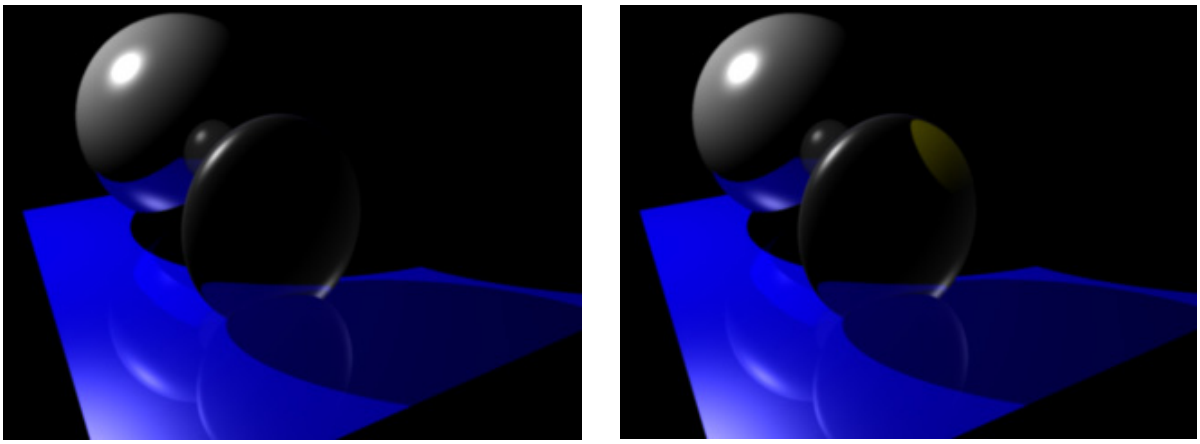


Figure 15 Hidden Geometris Visible in Refractions Off vs. On

6.7. Hidden Geometries Produce Reflections

Turning on this option results in visible geometries being also visible through reflections off of objects that are excluded by pass contribution maps, but present in the current render layer. In order to illustrate the feature, the pass contribution map of the example scene was edited to exclude the gray reflective sphere.

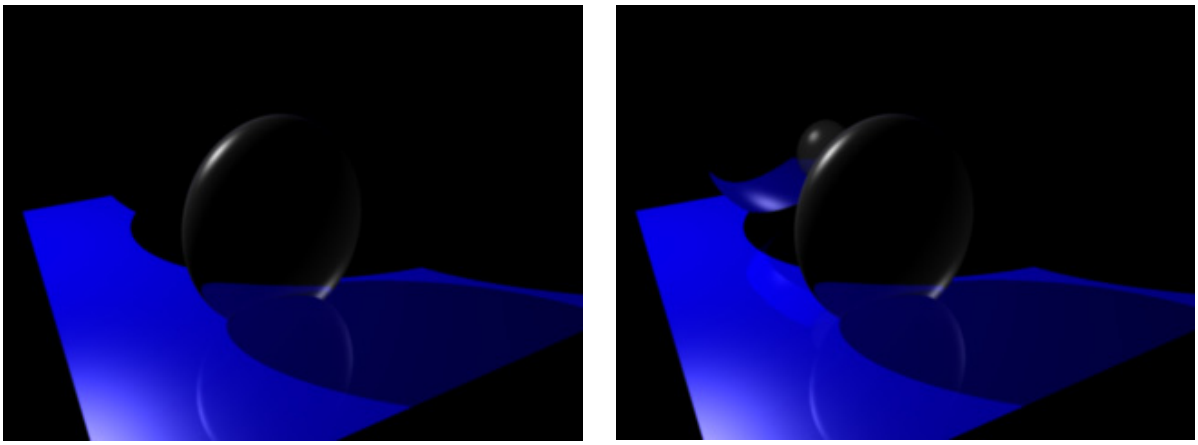


Figure 16 Hidden Geometries Produce Reflections Off vs. On

6.8. Hidden Geometries Produce Refractions

Turning on this option results in visible geometries being also visible through refractions in objects that are excluded by pass contribution maps, but present in the current render layer. In order to illustrate the feature, the pass contribution map of the example scene was edited to exclude the refractive ellipsoid.

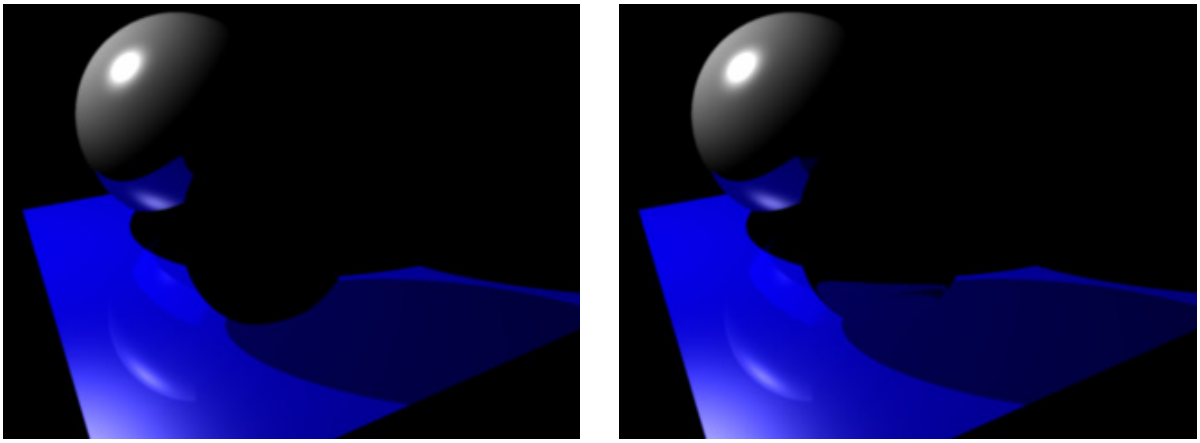


Figure 17 Hidden Geometries Produce Refractions Off vs. On

6.9. Minimum Reflection Level

This is the minimum number of reflections a ray needs to have traversed before contributing to the render pass. A value greater than zero means that no objects are directly visible to eye rays. This option can be used to help isolate reflections.

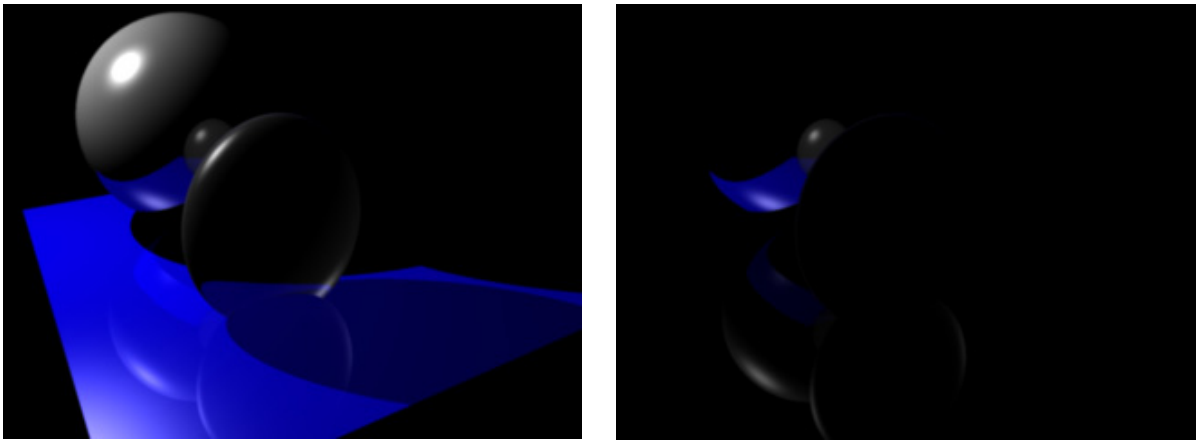


Figure 18 Minimum Reflection Level 0 vs. 1

6.10. Maximum Reflection Level

This option has the same effect as the Reflections parameter under the Raytracing section of the Quality tab of the Render Settings window. The difference is that the global setting in the Render Setting limits the trace depth, which affects all passes, including the Master Beauty, which may have a significant impact on render time. On the other hand, the render pass setting only affects that specific pass, and it does not override the global setting, so setting the value higher than the global setting has no effect.

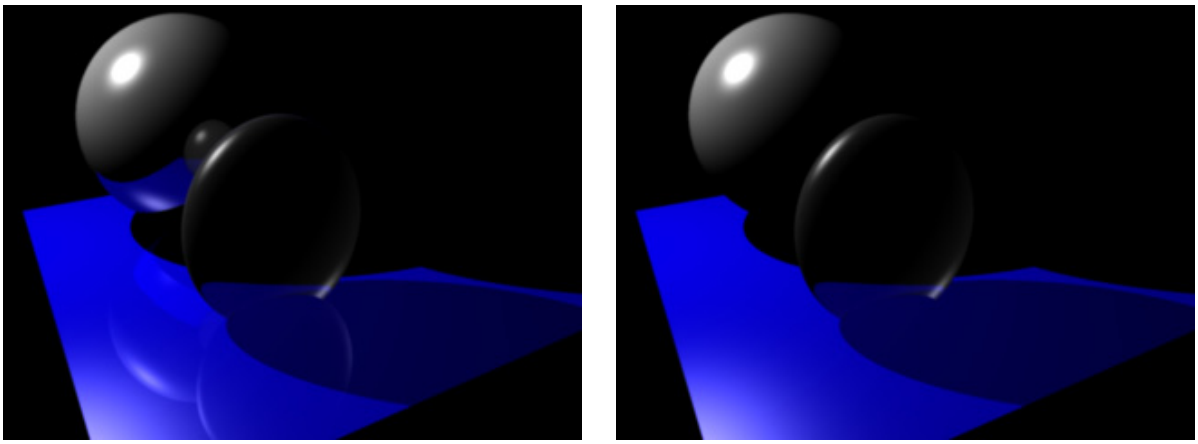


Figure 19 Maximum Reflection level 10 vs 0

The right-hand-side images from Figure 18 and Figure 19 are complementary. Adding them together produces an image similar (up to machine precision) to the original beauty passes (left-hand side images), that contain all reflection levels.

6.11. Minimum Refraction Level

This is the minimum number of refractions a ray needs to have traversed before contributing to the render pass. A value greater than zero means that no objects are directly visible to eye rays. This option can be used to isolate refractions.

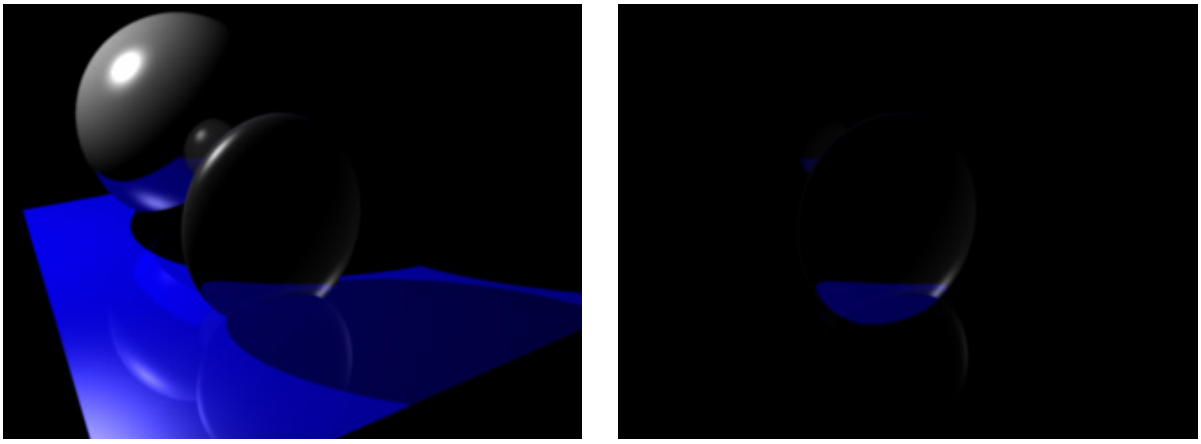


Figure 20 Minimum Refraction Level 0 vs. 1

In the bottom and right of the right-hand-side image of Figure 20, there is a faint sliver that represents a reflection. This is the reflection of refracted light, which technically meets the criteria of a minimum refraction level of 1. In order to isolate pure refraction, the max reflection level must be set to 0. Unfortunately, the current framework provides no means to discriminate between a reflection of a refraction, and a refraction of a reflection, which would both have a reflection level of 1 and a refraction level of 1.

6.12. Maximum Refraction Level

This option works essentially the same way as the Maximum Reflection Level, but with refractions.

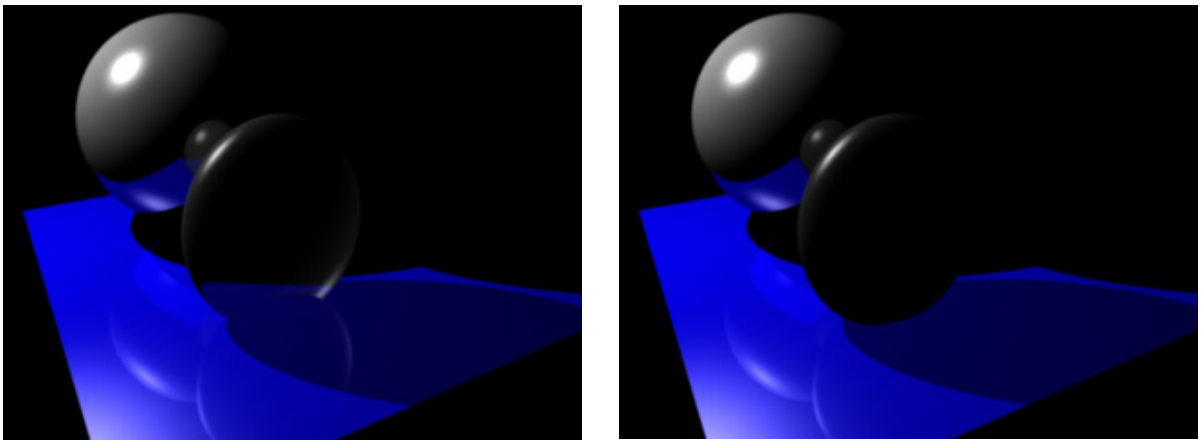


Figure 21 Maximum Refraction Level 10 vs. 0

Just as in the reflection case, the right-hand-side images from Figure 20 and Figure 21 are complementary and can be added together to reconstitute a full beauty pass.

7. Render Pass Presets

When creating a render pass with the Passes tab of the Render Settings window, the user must chose the type of pass from a list box. Each item in that list box is a preset that comes from a preset file located in *maya_install_directory*/presets/attrPresets/renderPass. These files can be edited and new ones can be added in order to provide a customized render pass palette. It is recommended to backup the contents of this directory before editing them.

7.1. Editing Default Values

One important advantage of editing render pass presets is that the default settings of newly created render passes match production requirements. For instance, most render passes are in 16-bit float format by default. If a project requires 32-bit float renders, then it may be worthwhile to make that change in the presets to avoid rendering in the wrong format by error during production. Setting a new default for a parameter is achieved with the blendAttr mel command. For example, adding the line

```
blendAttr frameBufferType 512;
```

The built-in attributes for renderPass nodes that are relevant for presets are: passID (string), frameBufferType (integer), numChannels (integer), filtering (Boolean), passGroupName (string). The passID attribute should never be changed directly with blendAttrString. Instead, the passID must be set using the setRenderPassType command, which takes care of creating all the dynamic attributes that are relevant for a given pass type. The preset may then assign default values to the newly created dynamic attributes.

The frameBufferType attribute is the only one that is not straightforward because the data types are encoded as integers. The currently accepted values are:

Frame Buffer Data Type Encoding

Type	Value
8-bit unsigned integer	1
16-bit unsigned integer	2
32-bit unsigned integer	4

64-bit unsigned integer	8
8-bit signed integer	16
16-bit signed integer	32
32-bit signed integer	64
64-bit signed integer	128
16-bit floating-point	256
32-bit floating-point	512
64-bit floating-point	1024
1 bit	2048
Other	4096

The formats that are supported may depend on what renderer is being used, what file format is being rendered to, and the semantic of the render pass type. When an unsupported value is specified, the format reverts to the default for the given pass type.

7.2. Adding Presets

New render pass presets may be created at will by adding new preset files to the renderPass preset directory. Changes to the contents of that directory take effect immediately without restarting Maya.

All new render pass preset files should follow the following format:

```
startAttrPreset( "renderPass" );
global string $gAEAttrPresetCurrentTarget;
setRenderPassType -defaultDataType -type "type ID"
$gAEAttrPresetCurrentTarget;
blendAttr statements
endAttrPreset();
```

The type ID string must be one of the currently supported render pass types, registered with the render pass registry. A list of available render pass types can be obtained by typing the following command in MEL:

```
renderPassRegistry –supportedRenderPasses;
```

Maya has a number of built-in render pass types, but additional types may be added by 3rd party renderer plug-ins.

8. .mi File Representation

When rendering with mental ray, scene entities used to describe render pass behavior are translated into a representation that is understood by the shaders of the Maya base shader library. This data representation is exposed when the scene is exported to a .mi file, which allows users to edit the behavior manually with a text editor, or through user-written scripts.

8.1. File Export Options

It is possible to control which pieces of render pass data gets translated through some simple export options. When exporting a mental ray assembly, none of the render pass information is exported.

The first option related to render passes is *Export pass contribution maps*, which is on by default. Turning this option off causes the pass contribution maps to be left blank during translation, which causes all lights and all geometry instances in the current render layer

to be rendered in all render passes. The pass contribution maps are encoded as a per-instance parameter in the mental ray scene description language, and refer to user frame buffers that are specific to the current camera. Therefore, it may be desirable to exclude pass contribution maps from translation when exporting a scene destined to be merged with other scenes, which may not have the same render passes defined.

The *Export pass user data* option determines whether or not the frame buffer data blocks are exported. They are user data blocks with the ":fbdata" suffix that encode the configuration of each render pass. There is one frame buffer data block for each renderable camera in a scene.

8.2. Render Pass Translation

In this context, the term *translation* refers to the process of converting the Maya scene representation into the mental ray data model. With mental ray for Maya, there are two main translation paths: exporting to a .mi file (for use with mental ray standalone), and translation through the mental ray API (for rendering directly from within Maya, including batch rendering). The API translation path is of little interest to most users because it is not accessible to the user. In this section, we shall focus on the .mi file representation, which can easily be edited by the user if desired. Even for users who do not use mental ray standalone, exporting .mi files can be very useful for debugging renders that do not behave as expected. The less technically oriented reader may want skip this entire section.

Note: The schemas described in this section reflect data representations used in Maya 2009 through Maya 2011. There is no guarantee that future versions of Maya will maintain compatibility with these schemas. Also, there is no guarantee that future versions of corresponding data representations will be user-accessible or documented. Therefore, there is no expectation that user-written software tools or scripts that make use of these schemas will be forward-compatible or even portable to future versions of Maya.

8.2.1. The Frame Buffer Data Block

All of the parameters specified in Maya render pass nodes result in a frame buffer data block.

In an exported .mi file, that data block takes the following form:

```
data "perspShape:fbdata"
"adskFrameBufferData" (
"magic" 1178760550,
"nonMaterialPassFrameBufferNames" [
       (…)
],
"frameBufferInfo" [
       (…)
],
"frameBufferTypeCounts" [
       (…)
]
)
```

The magic field is used for data structure identification, and must always be set to 1178760550.

The nonMaterialPassFrameBufferNames member is an array of strings which are the names of the frame buffers to which pass shaders render non-material passes. All pass shaders (adskPassCameraDepth for example) have an integer parameter named frameBufferNumber to identify the pass that the shader writes to. This number is in fact an index into the nonMaterialPassFrameBufferNames array, which is looked-up during shader initialization. This indirection is necessary because the name of the frame buffer

that is written to may vary from one camera to another. For example, in the case of stereo rendering, where both the right and left eye images are stored in the same .exr file, the two must have distinct buffer names in order to avoid a conflict. The name switching is handled through the per-camera frame buffer data block.

The frameBufferInfo member is an array of data structures used for describing the properties of material render passes. The parameters of the data structure are direct translations of the render pass parameters specified on the Maya renderPass node.

The frameBufferTypeCounts member is an array of integers that specifies the number of frame buffers that belong to a given material render pass type. The size of the array is always equal to NUMBER_OF_PASS_TYPES, declared in adskRenderPassTypes.h (in the shader SDK). The PassTypeID enumerated type provides the index into frameBufferTypeCounts corresponding to a given pass type.

8.2.2. Pass Contribution Map Encoding

The behavior of pass contribution maps was explained in section 4.1. Now, let us look at how the pass contribution map connections in the Maya scene are translated for rendering with mental ray. The Maya scene model is designed to simplify the representation of DAG-object-to-pass relationships by consolidating them through pass contribution maps. These relationships are thus broken-down in to two levels: DAG object to PCM, and PCM to render pass. This design is more scalable (vs. direct DAG object to pass connections) and simplifies the management of these relationships since it generally requires much fewer plug connections. The mental ray scene representation, on the other hand, is designed for rendering performance. During the scene translation process, the pass contribution maps are resolved in order to obtain the list of passes that receive a contribution on a per-shape instance basis. This information gets encoded into the mental ray scene as an instance data parameter. In an exported .mi file, a typical shape instance with a pass encoding looks like this:

```
instance "nurbsSphere1" "nurbsSphereShape1"
      light "exclusive" []
      material ["initialShadingGroup"]
      hide off
      shadow 0
      transparency 0
      reflection 0
      refraction 0
      caustic 3
      globillum 3
      transform
            1. -0. 0. -0.
            -0. 1. -0. 0.
            0. -0. 1. -0.
            0.224519 0. 0.102098 1.
      (
            "passEncoding" "173A"
            )
end instance
```

The passEncoding instance data field is a hexadecimal encoding of a bitstream. Each bit represents the on/off state of the instance for a given render pass. The bit position is the ordinal position of the corresponding render pass in the frameBufferInfo array of the frame buffer data block. For example, to determine whether the second render pass described in the data block receives a contribution from a given instance, one must look at the value of the second bit of the passEncoding string. If the passEncoding string is not specified, the instance contributes to all render passes. The hexadecimal string is in big-endian digit order. In other words, the character position from the beginning of the string for the N^{th} bit is given by the integer part of $N/4$. The passEncoding string can be longer than the

number render passes in the frameBufferInfo array;the extra bits are used for non-material render passes. The bit position used for a non-material render pass is specified by the encodingIndex parameter of the associated pass shader call.

In C++ code, the *contributesToPass* function from the Shader SDK can be used to interpret pass encoding strings.

8.2.3. The Options Block

The scene options block does not store any options for the render pass system, but there are two elements that need to be present in order for render passes to render correctly: the traversal shader, and the state shader. These are automatically taken care of by the Maya scene translator when exporting a .mi file, and they must not be removed. The traversal shader is specified as such:

```
traversal adskTraversal
```

The traversal shader is used for transporting instance parameters during DAG traversal. Since the mental ray for Maya translator flattens the DAG, this shader does not actually perform any traversals, but its declaration provides the structure for instance data (the passEncoding string).

The state shaders should be specified as such:

```
state [
      "maya_state" (
            )
      ,
      "adskFrameBufferState" (
            )
      ]
```

The maya_state shader manages general-purpose state used to relay data between light shaders, material shaders, and the shading engine. It is necessary for using built-in Maya shaders, even without render passes. The adskFrameBufferState shader manages data structures that store configuration information, as well as intermediate data used specifically by the render pass framework. All the function and data structures for accessing the structures are declared and documented in adskFrameBufferState.h (in the shader SDK).

Another item in the options block that is important for render passes is:

```
"contrast all buffers" on
```

This option affects the behavior of the contrast criterion for adaptive sampling. Without this option, mental ray will refine sampling when a sharp contrast between neighboring samples is detected in the main frame buffer (the Master Beauty pass). This may lead to poor sampling (i.e. aliasing, jaggies) in other passes, especially when using pass contribution maps. By turning on this option, mental ray performs the contrast test on render passes that store color data.

8.2.4. Material Definitions

All material render passes are managed directly by the material shaders, the shading engine and the state shader, so they have no impact on material definitions. However, non-material passes are rendered by individual shaders that are executed sequentially, after the main shading network. This sequence is represented as a material shader list in the material definition. For example:

```
material "myShadingGroup"
      "adskMayaShadingEngine" (
            "surfaceShader" = "lambert1.outColor",
```

```
            "cutAwayOpacity" 0.,
            "customShader" off
            )
      "adskPassCameraDepth" (
            "frameBufferNumber" 2,
            "encodingIndex" 11,
            "holdout" off,
            "useShadingEngineThreshold" off,
            "transparencyThreshold" 0.,
            "remap" off,
            "znear" 0.,
            "zfar" 1000.,
            "minbuffer" 0.,
            "maxbuffer" 1.
            )
      "adskPassMotionVector2D" (
            "frameBufferNumber" 4,
            "encodingIndex" 13,
            "holdout" off,
            "useShadingEngineThreshold" off,
            "transparencyThreshold" 0.
            )
      shadow = "lambert1:shadow"
end material
```

The call order of the pass shaders is not important, as long as they each contribute to different frame buffers. In order to respect proper compositing order, it is important that the pass shaders are called after the shading engine, which is responsible for casting transparency rays.

8.2.5. Shadow Shaders

Because of pass contribution maps, it is possible for an object to cast shadows in some render passes and not in others. As explained in 3.1.5, this only works with direct illumination. In order to support independent shadow casting on a per-pass basis, specially designed shadow shaders are necessary. Two such shaders come with Maya: adskMayaShadow, and adskMayaFastShadow. The difference between the two shaders is that adskMayaFastShadow only supports fully opaque shadowing. When translating built-in base materials, Maya automatically chooses the appropriate shader based on whether or not the surface is opaque. However, these shadow shaders are only used automatically with materials from the standard Maya shader library. Therefore, **non-Maya shaders will not produce direct illumination shadows in render passes**.

Tip: To workaround this problem so that non-Maya shaders cast direct illumination shadows in render passes, use the surfaceShader shader as an intermediate between the material shader and the shading engine. If the material shader is a shader or phenomenon that has built-in render pass support (mia_material_x_passes, for example), donot forget to turn off render pass contributions from the surfaceShader shader to avoid double contribution to beauty passes.

8.2.6. The Camera Block

The camera block contains the frame buffer declarations for each render pass. They are specified per-camera because that is how the mental ray scene representation works. However, the Maya data model does not allow the render pass configuration to change from one camera to another; it can only change between render layers. The only things that can change between cameras within the same render layer are the output file names and the frame buffer names. The actual render pass list cannot change, especially considering the fact that the pass contribution encodings do not differentiate on a per-camera basis.

The camera block also contains a reference to its corresponding frame buffer data block.

8.3. Using Render Passes with Render Proxies[7]

The use of render proxies is great for improving viewport performance, scene translation time, render time, and memory consumption with complex scenes. Unfortunately the use of render proxies imposes a translation barrier between the contents of the proxy and the rest of the scene, since the proxy is a pre-translated entity. In practice, this means that it is impossible to obtain any interaction between proxy contents and elements of the parent scene that would require any type of explicit connection, such as light linking, and material assignment. As far as render passes are concerned, material render passes defined in the parent scene are picked-up by the shaders in the render proxy because they do not require any explicit connections. On the other hand, non-material passes are problematic because they affect the translation of material definitions. The problem is that the pass shaders for the non-material passes defined in the parent scene cannot be tagged-on to the material definitions in the render proxy.

Also, pass contribution maps are ignored during render proxy export. There is no guarantee that the render pass configuration in the parent scene will be consistent with the render pass configuration of the scene from which the proxy was exported. Therefore importing render pass encodings would be unreliable, which is why PCMs are not supported with render proxies. However, the placeholder geometry in the parent scene may be associated to pass contribution maps. **The PCM memberships of the placeholder geometry will propagate to the contents of the render proxy**.

9. Render Pass Naming

The render passes are named according to their node names. Their names may be used in the construction of file names and frame buffer names.

9.1. File Naming Mechanisms

In the Common tab of the Render Settings window, the *File name prefix* field specifies the string format used for constructing the first part of image file names (excluding the frame number and format extension). Special tokens can be used in order to insert render pass information strings into the file naming scheme:

- <RenderPass>: The name of the render pass node. If this token is omitted in a scene that renders multiple render passes per layer and per camera, then Maya automatically creates a separate subdirectory for each render pass to avoid file name clashes
- <RenderPassType>: This is a unique identifier that identifies the render pass type. This token can be very useful for setting-up pipeline automation scripts that may perform different tasks depending on render pass type, which may be identifiable from the file name.
- <RenderPassFileGroup>: Render pass nodes have a Pass Group Name attribute, which specifies the value for this token. This is a custom identifier, which, unlike the render pass name, is not necessarily unique in the scene. The token is meant to be used for grouping purposes. For example, the following file name prefix would group render passes into subdirectories according to their group name: `<RenderPassFileGroup>/<RenderPass>`

[7] Render Proxy is a renderer-neutral term used in Maya to refer to mental ray "assemblies", or an equivalent technology in a different renderer. In the current context, *render proxy* and *mental ray assembly* can be used interchangeably.

9.2. Frame Buffer Naming (for OpenEXR® files)

In the Common tab of the Render Settings Window, the Frame buffer naming options are grayed-out unless a multi-channel file format, such as OpenEXR, is selected. Frame buffer names are generated during scene translation, and used as identifiers for frame buffers. mental ray uses the frame buffer names as channel names for storing multiple images in the same .exr file.

The standard rendered file naming logic in Maya automatically creates subdirectories to help resolve file name conflicts. When rendering to .exr, these subdirectories are only created to help resolve clashes between render layers. Image name differentiation based on cameras and render passes is enforced in the frame buffer naming. When Frame Buffer Naming is set to *Automatic*, the following default format is used:

<RenderPassType>:<RenderPass>.<Camera>

When Frame Buffer Naming is set to *Custom*, and the Custom Naming String does not contain the <Camera> and <RenderPass> tokens, Maya may automatically override the Custom Naming String in order to help resolve name clashes. For separating groups of render passes into different .exr files, the <RenderPassFileGroup> token should be used in the file name prefix.

The current version of OpenEXR (1.6.1) limits channel names to 31 characters. Because of this limitation, frame buffer names often need to be truncated. A simple truncation is not sufficient to ensure that names remain unique (in order to avoid clashes). To help resolve the problem, Maya creates truncated names that end with a CRC-32 hash of the un-truncated string. This is not ideal because the name becomes unintuitive. In extremely rare circumstances, this mechanism may still fail to produce a unique name, which may lead bad or missing buffers. This problem can be avoided by using short camera and render pass names. Exporting a .precomp file (using the *Export Pre-Compositing* item in the *Render* menu) can help sort-out truncated channel names. The precomp file is a simple Python module that defines, among other things, the hierarchy of cameras, layers and render passes, and provides the image file and image buffer names for all rendered image streams.

10. Transparency

This chapter is a primer for further discussions on compositing. For a long time, computer graphics practitioners have been using alpha compositing to achieve image combination effects. The best-known combination operation is the alpha-blending equation (also known as the *Normal* blending mode):

$$Result = (1 - \alpha)Value_{background} + \alpha \cdot Value_{foreground}$$

This equation is commonly used in 2D image compositing for applying mattes, where the alpha value represents the pixel coverage fraction of the foreground layer.

Because it is convenient and compact, alpha-blending has also been widely used to produce transparency effects. However a single alpha channel is not a very good model for representing real-world transparency phenomena. In the real world, transparent objects transmit different light wavelengths in varying proportions. Therefore, it is more appropriate to model transparency on a per-color channel basis. A very common model used in computer graphics today is one alpha value per color channel. This model is usually referred-to as a transparency color. This is the model used in the Maya base surface shaders. In Maya, colors have no alpha component, which avoids confusion between actual transparency and coverage masks.

Transparency represents the fraction of light that is transmitted through the surface. This is given by 1-alpha, where alpha represents opacity, or the fraction of light that is occluded by the surface. In the alpha blending equation, the foreground is multiplied by opacity,

which is not a physically correct way to model most types of semi-transparent surfaces. This problem is elaborated later in this chapter.

10.1. The Meaning of *Premultiplied*

In digital compositing lingo, the term *premultiplied* is used to designate graphics images that have already been multiplied by their alpha values (i.e. already masked by their mattes). This means that the alpha blending equation becomes:

$$Result = (1 - \alpha)Value_{background} + Value_{foreground}$$

Maya renders premultiplied images. Technically speaking, the rendered pixel values are never actually premultiplied *per se*. The baked-in alpha multiplication is actually a result of sample filtering, which is necessary for producing anti-aliased images. Many compositors that are used to working with live-action footage tend to assume that if an image is premultpilied, it must be because someone premultiplied it, which is a logical assumption, but it is actually often not the case with computer generated images, since they can be rendered with no background (*i.e.* pre-matted). This is not a problem since most compositing applications have built-in functionality (options, tools, or special blending modes) for dealing with premultiplied images.

Furthermore, real-world tinted transparent objects cannot be modeled by the first equation (where the foreground is multiplied by alpha) because it implies that the surface's reflectance (diffuse and/or specular) is capped by its opacity, which is incorrect. For example, according to alpha-blending logic, if a surface only transmits red light, its RGB opacity could be (0,1,1), meaning that the surface could only reflect green and blue. Many physical materials disprove this model, such as glossy light-filtering materials. Therefore, compositors need to break away from the concept of multiplying foreground layers by opacity (or alpha), because that paradigm cannot always yield a physically-correct simulation of transparent objects. Any attenuation of the foreground color due to transparency should normally be baked-in to the rendered result by the surface shader.

The implications of premultiplication in compositing are explained in greater detail in 11.2.

10.1.1. The *Premultiply* Rendering Option

mental ray for Maya has a *Premultiply* option in the *Framebuffer* section of the *Quality* tab of the render settings. This option is not truly a premultiplication option, it should be interpreted as: "do not postdivide". When the option is turned off, all frame buffer color values are divided by their respective alpha values. This option must be used with caution since the alpha channel may not always be right for expressing transparency, as explained above.

Also, turning off the premultiply option may generate artifacts if the background is anything but pure black. Another inconvenience is that this is a global mental ray setting, so it always applies to all render passes. It can, however, be turned on and off on a per-layer basis using the render layer override mechanism. When in doubt, it is best to have the option turned on, which is its default state. There are always alternate ways of dealing with this issue properly in compositing.

One noteworthy case where it is useful to turn off *Premultiply* is for rendering images that are intended to be used directly as textures (e.g. texture baking), without any intermediate compositing.

10.2. Alpha Channels of Render Passes

In the mental ray for Maya render pass system, the notions of transparency and coverage are kept separate: the alpha channel is generally used for storing coverage masks, while physical opacity is expressed by the Opacity render pass type. Material opacities (or

transparencies[8]) cannot be rendered to an alpha channel since they require three channels in order to be fully represented. In traditional compositing workflows, the alpha channel is usually used as the compositing matte, which works fine for opaque objects since there is no real distinction between coverage and opacity in this case. However, the composition of light filtering materials requires an opacity matte. The alpha channel of render passes is not affected by transparency, except for the Master Beauty (for backwards compatibility). This disconnect is sometimes perceived as a bug by Maya users because the transparency inputs of material shaders are often used to receive the coverage mask (alpha channel). This is a valid workflow, but it does not always generate the matte that the user expects in the render pass alpha channel. The transparent areas of the texture are rendered with alpha values of 1.0 (fully opaque). This is expected because the texture's alpha, although it originally represented a coverage mask, was reinterpreted by the surface shader as an opacity mask. The *Opacity* render pass type helps produce the desired matte in such cases.

10.3. Transparency vs. Refraction

The real-world phenomena of transparency and refraction are almost the same; the nuance is that the term refraction implies that light rays are bent. In mental ray, transparency and refraction are treated very differently because there are important implications. Transparency is the special case of refraction where the refracted ray is the continuation of the incident ray. Because the transmitted ray is perfectly aligned with the incident ray, it is easier to re-create the phenomenon in compositing using an opacity matte. On the other hand, the bending of rays that happens in refraction is nearly impossible to recreate accurately in compositing because doing so requires knowledge of three-dimensional aspects of the scene. For this reason, refractive surfaces are represented as fully opaque in the opacity render pass.

Also, in the mental ray for Maya shader ecosystem, refraction rays are cast by the surface shaders, while transparency rays are cast by the shading engine. This may seem like an insignificant implementation detail, but it has important repercussions. The shading engine needs to take control of transparency casting in order to apply the transparency threshold criterion, compute cumulative mattes, and force transparency rays that are required for render passes (due to PCMs). This will cause problems with certain third-party or user-written shaders that were not necessarily designed to work with the Maya shader framework. Such shaders typically cast their own transparency rays. In that case, the material is treated as opaque by the shading engine. This scenario prevents the shading engine from producing correct transparency mattes, and it is not-optimal because the shading engine may still cast the same transparency ray in order to satisfy render pass visibility considerations expressed by PCMs.

10.4. Applying Transparency to a 3rd Party Shader

Although generic mental ray shaders don't comply with the Maya shading framework, they can still be used to produce transparencies in a Maya-friendly way by bootstrapping them to a surfaceShader shader. The idea is to let the surfaceShader node manage the transparency. This method can even be used to apply transparency to shaders that otherwise don't even support transparency, like the mib_illum_* shaders[9].

The following example shows how to apply a texture as an alpha transparency map to a shader from the mib library.

[8] Transparency and opacity are intrinsically linked because they are complements of each other.

[9] The mib_illum_* shaders are the surface shaders of the mental image base shader library. These shaders and their source code are distributed with Maya, but they do not strictly comply with the Maya shading framework.

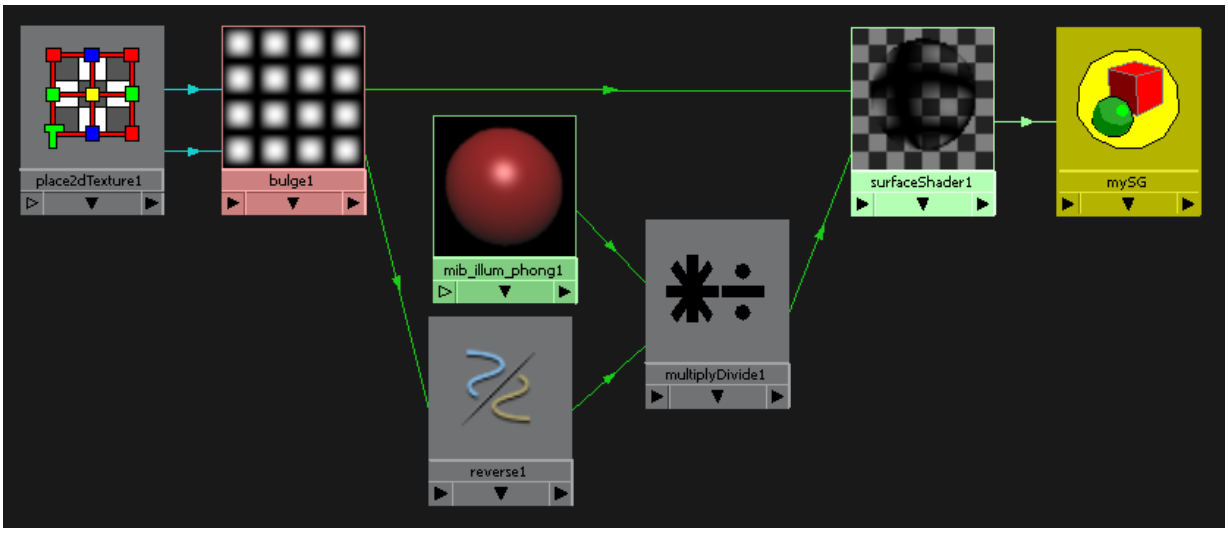


Figure 22 Applying Transparency with surfaceShader

The outValue of the bulge shader is connected to the outTransparency of surfaceShader1, while the complement is used to modulate the result of mib_illum_phong1, which is equivalent to an alpha blend with the background. This shading network allows the shading engine to produce an opacity render pass that represents the material's opacity.

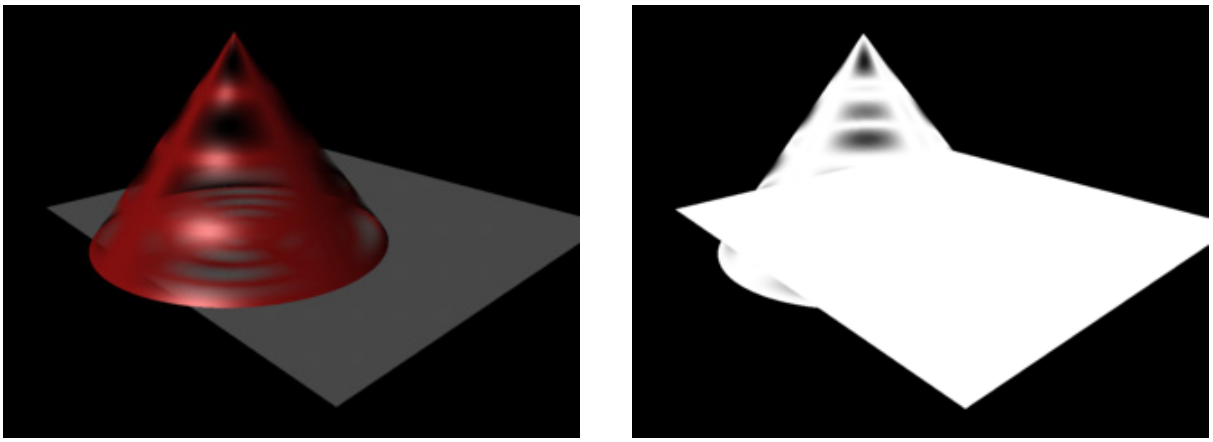


Figure 23 Master Beauty (left) and Opacity (right)

In Maya 2009 and 2010, the opacity of superposed objects was not composited between superposed surfaces in *Opacity* passes. The above images were rendered with Maya 2011, which was improved to perform opacity compositing.

According to the principles explained in 10.1, we have just committed heresy with the above example by multiplying the foreground by the opacity value. This would have been fine had the transparency texture represented a coverage mask. For argument's sake, let us assume that the rendered surface is in fact semi-transparent. In order to obtain a result that is more physically plausible, let us assume the hypothesis that the material we are modeling is clear semi-glossy plastic, injected with red pigment that is responsible for the bulk of the diffuse reflection and occlusion. The transparency mask represents pigment density. This material could be modeled as follows.

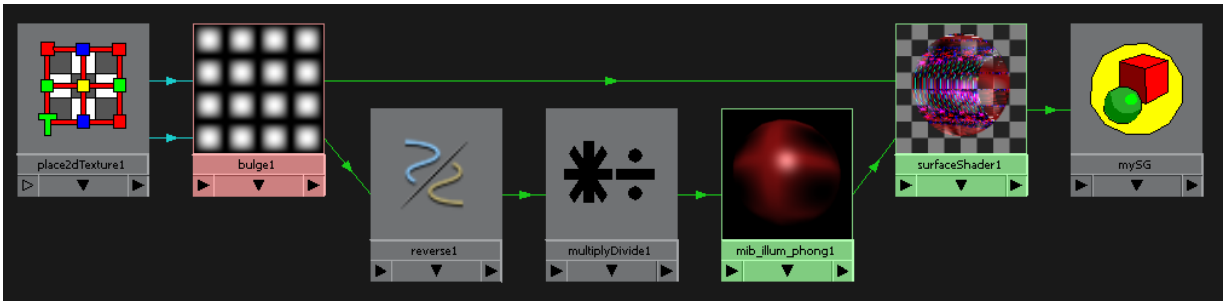


Figure 24 Semi-transparent Surface Using mib_illum_phong

The swatch for surfaceShader1 shows a corrupted image in this specific case, but the scene still renders fine. The output of multiplyDivide1 is connected to the diffuse input of mib_illum_phong1. The secondary input of multiplyDivide1 is set to dark red, which is the pigment color. The outputs of bulge1 and mib_illum_phong1 are respectively connected to outTransparency and outColor of surfaceShader1.

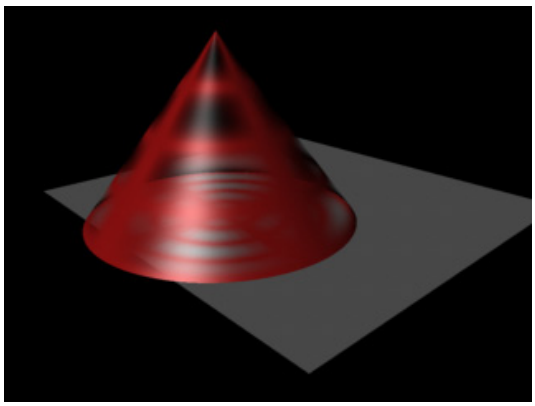


Figure 25 Semi-transparent Surface Render Result

This example shows the specular highlight having an additive effect with the background seen through the transparent portions of the cone's surface, as should be the case. This result is similar to the behavior of Maya's Phong material shader. The opacity remains unchanged from Figure 24, since the specular highlight is non-occluding. This example supports the case made in 10.1 about the necessity of having the effect of transparency pre-baked into the render for compositing purposes. If that were not the case, the compositor would have no way of masking the diffuse color without also masking the non-occluding glossy reflections – unless of course the diffuse and specular passes were both rendered, but why do more work than necessary?

11. Compositing Guidelines

11.1. Basic Compositing Arithmetic for Combining Passes

Many of the material pass types consist of components of the overall BRDF[10]. These passes can be used as a decomposition in order to tweak material properties at the compositing stage. The base material shaders all use the same basic compositing equation:

$$Beauty = (Diffuse + Translucence + Scatter + Indirect) \cdot (White - Transparency) + Specular + Reflection + Refraction + Incandescence + Ambient$$

Re-composing a layer from component passes can be useful for balancing a material's BRDF in a compositing software application, for example, applying a color-correction to the diffuse color of an object, without affecting reflections, and without having to re-render.

[10] A BRDF, or Bidirectional Reflectance Distribution Function, is a mathematical function that defines how light is reflected by a surface. The Maya base shaders implement analytical BRDF models that are based on a set of standard components. The components represent various optical phenomena, as well as different computational variants of optical phenomena (e.g. direct vs. global illumination).

Some shaders may add some effects that are not accounted for in any component passes. Such effects are only visible in beauty passes. For example, this is the case of the foam effect in the ocean shader. It is possible to isolate custom effects by subtracting the computed recomposition from a beauty pass.

It is also possible to use an even deeper decomposition than the above equation by applying the following identity:

$$Diffuse = DiffuseMaterialColor \cdot DirectIrradiance$$

The *Shadow* pass type represents the difference in color values with and without shadow computations. Shadow passes are only capable of representing direct illumination shadows (ray-traced or depth map shadows). Indirect shadows from global illumination or final gathering are implicitly expressed in the *Indirect* pass and cannot be separated out. There are two types of shadow pass: raw and regular. The raw shadow represents the difference in direct irradiance, while the regular shadow is the difference in the final shading result.

$$Shadow = BeautyNoShadow - Beauty$$

$$RawShadow = DirectIrradianceNoShadow - DirectIrradiance$$

$$\therefore Diffuse = DiffuseMaterialColor \cdot (DirectIrradianceNoShadow - RawShadow)$$

11.2. Compositing Scene Partitions

It is a common practice to decompose a scene in different groups of objects that are rendered separately and recombined in compositing. This allows the compositing artist to manipulate different partitions of the scene independently. This section discusses the problems encountered during the process of recombining the partitions, and in particular dealing with semi-transparent, reflective, and refractive scene elements, which can be tricky.

Traditional 2D image compositing usually uses alpha blending, or another form of blending to combine a foreground image with a background image, weighted according to an opacity mask. Such operations can be performed sequentially to composite a stack of image segments. As demonstrated in section 10, it can sometimes be challenging to generate transparency mattes that accurately represents the transparency modeled by the shader, especially with non-Maya-compliant shaders. Depending on the production application, different compositing strategies may be appropriate.

11.2.1. Pre-Matted Compositing

The basic premise of pre-matted compositing is to have the occlusion mask baked-in to the pixel color value. In other words, each image in the compositing stack is premultiplied. This way, no mattes are required and the compositing operation is no more than a simple addition.

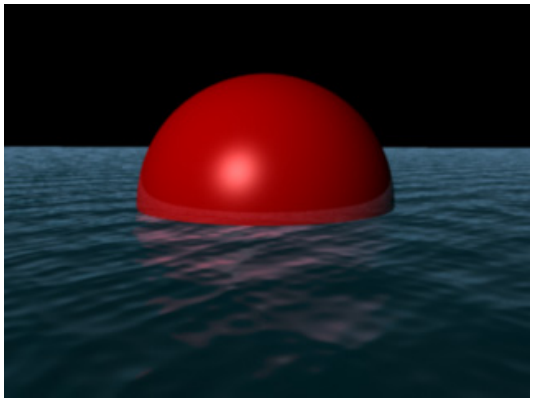


Figure 26 Buoy and Ocean Master Beauty Pass

Producing a compositing stack for a pre-matted scene decompositionis as simple as creating an array of beauty passes with associated PCMs that represent the scene partitions for each pass. To make the image pre-occluded, the *Hold-out* option must be turned on for all passes. In order to capture all the reflections and refractions between objects in different partitions, it is important to either:

a) turn on *hidden objects visible in reflections* and *hidden objects visible in refractions*, or
b) turn on *hidden objects produce reflections* and *hidden objects produce refractions*.

Option a) is good for manipulating a scene partition along with the reflections and refractions that are cast by the objects in that partition. Option b) allows the user to

manipulate a partition along with the reflections and refractions of the objects in that partition cast by objects in other partitions. Using b) may sound less intuitive, but it actually has very practical use cases, such as re-coloring. For example, take a simple scene of a buoy floating in the ocean and its partitions:

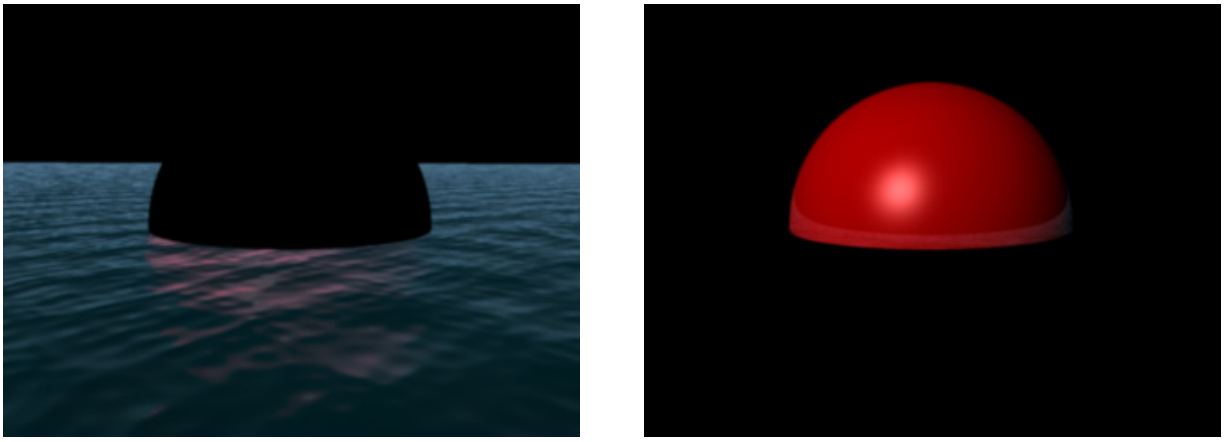


Figure 27 Buoy and Ocean Partitioned with Method a)

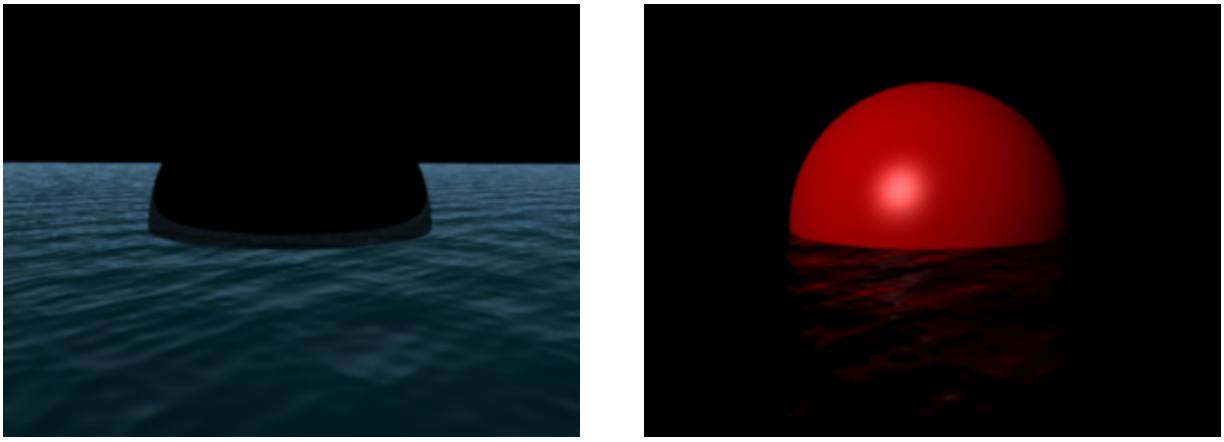


Figure 28 Buoy and Ocean Partitioned with Method b)

With both methods a) and b), the Master Beauty is equal (up to machine precision) to the sum of the partition images.

In the case where the buoy is to be re-colored in the compositing stage, the color of the buoy's reflection in the water would need to follow suit. With method b), achieving that is very simple. It is just a matter of color correcting the buoy image and adding the result to the ocean image. Also, it should be noted that the reflection of the ocean in the buoy would not be affected by the re-coloring.

Another interesting advantage of pre-matted compositing is that image addition is commutative. In other words, the order in which the images are added together is not important. On the other hand, matte-based compositing requires the image to be composited in depth order. The problem of depth-sorting scene partitions is often trivial, but there are cases where it is not possible. An example of a problematic case is with volume rendering when a semi-transparent solid object is inside a semi-transparent volume. Depth-based compositing of the solid object with the volume would be problematic because each partition is an occluder of the other. The commutative property of pre-matted compositing avoids this problem entirely.

The main limitation of pre-matted compositing is that, because the occlusions are baked-in, the relative disposition of the scene partitions is not allowed to change. The reason is

that the images need to remain perfectly lined-up for the occlusions to match the positions of their respective occluders.

Another important limitation is that this technique is only convenient to use for compositing images that were rendered from the same virtual scene; otherwise it may be difficult to pre-occlude the different scene partitions. For compositing over live-action footage, one possible solution for pre-occluding the footage is to load the footage as a camera background image plane in Maya. With pre-matted compositing however, the back plate needs to be rendered in a separate render layer to avoid interfering with the other image partitions.

11.2.2. Standard Matte-Based Compositing

Using a matte means that occlusions are applied in compositing, which offers a higher level of editorial freedom to the compositing artist. The different partitions can be geometrically transformed in compositing, and still recombined correctly. To generate material render passes that are appropriate for matte-based compositing, one must turn off the *hold-out* option, and turn on *hidden objects visible in reflections* and *hidden objects visible in refractions*. If none of the objects in the scene have transparency, then the alpha channels of the render passes can be used as mattes. Otherwise, opacity passes should be used to generate correct transparency mattes. The mattes should also be rendered with the *hold-out* option turned off.

This editorial freedom comes at a cost though. If the different partitions are transformed relative to each other, then many optical interactions between objects in different partitions (i.e. shadows, reflections, refractions, caustics, color bleeding, *etc*.) may be invalidated. The only optical effect that can be easily correctly reconstructed is straight-line transparency. However, it is possible to manipulate shadows, reflections and other optical effects by rendering them-out to separate render passes. This gives the compositor a chance to make-up for changes in object disposition in order to obtain seemingly correct results without having to re-render the scene. This process can be somewhat laborious. Also, it is typically less physically accurate than re-rendering, but it is often good enough to trick the audience.

Also, in conventional matte-based compositing, the partitions need to be composited in correct depth order. This requires some extra logistics. Sometimes, rendering a depth buffer per partition can be used to help resolve the problem and obtain correct results even with partitions composited in any order. This requires a compositing application that has three-dimensional capabilities (a.k.a. 2.5-D). Even so, anti-aliased edges and semi-transparent surfaces may not render correctly, depending on the rendering technique used by the compositing application. Autodesk® Flame® 2011 software, for example, uses a hardware rendering technique that, in most cases, requires layers to be composited in depth order to avoid edge aliasing and transparency artifacts.

11.2.3. Un-Pre-Multiplied Matte-Based Compositing

Compositing with un-pre-multiplied images provides an additional level of control to the compositor. Because the matte's masking is not baked-in, it becomes simple to produce additional effects in compositing: editing object transparencies, controlling edge feathering, adding extrusions. As explained in section 10.1, mental ray for Maya produces pre-multiplied images. A quick approach to getting un-pre-multiplied images is to divide them by their mattes. Compositing applications usually have a tool for performing this operation. Composite has the *Unpremultiply* tool, and Autodesk Flame has a divide option on layers in the *Action* module. The division approach has numerical stability problems when dealing with near-zero areas of the matte, especially when dealing with low dynamic range (non-HDR) integer-encoded images. Furthermore, it is impossible to recover the un-pre-multiplied color for completely masked (matte = 0) areas of the image.

An alternative method is to work with both a coverage matte and a transparency matte. The coverage matte is simply the render pass's alpha channel. The coverage matte can be used to un-pre-multiply an image that was rendered without transparency. This way,

the compositing artist can work with an un-pre-multiplied image that does not have any "transparency holes" in it. Rendering a render pass without transparency is as simple as disabling the *Use Transparency* option of the render pass. A downside of this approach is that rendering with transparencies disabled inevitably means that scene partitions with superposed semi-transparent objects may be problematic because of occlusions.

11.2.4. Shading Decompositions

An important detail to remember when tweaking transparency mattes is that physical transparency does not usually impact glossy reflections, including surface specularity. Therefore, when dealing with highly reflective materials, it can be appropriate to work with shading decompositions instead of beauty passes. A realistic material appearance can be reconstructed using the arithmetic from section 11.1. This way, if the compositing artist edits the opacity matte of a scene partition, the specularity and reflection components (which do not need to be un-pre-multiplied), would not be affected by the change in transparency.

11.3. Handling Environments and Backgrounds

The only render passes that capture environment color are beauty passes. The way to obtain a pure environment pass is to create an empty beauty pass. That sounds easy enough except that associating a render pass with an empty PCM has no effect because empty PCMs are ignored. There are two easy ways to get around this:

a) Create dummy object that is either completely transparent or outside of the camera's viewing frustum, and use a PCM that contains only that object. Attach a beauty pass to the new PCM and turn off the pass's *hold-out* option to reveal the environment without any occlusions.
b) Create a separate empty render layer.

Solution b) is cleaner from a workflow perspective, but it adds a lot of processing overhead associated with having an additional render layer. In most cases a) will result in better overall render-time.

Being able to produce an occluded background image is important for pre-matted compositing. There are render-pass-based and render-layer-based methods for achieving this. The traditional layer-based hold-out method (pre-Maya 2009) consists of creating a new render layer with a global material override to render objects as black occluders. This can be done using the surfaceShader shader. Dealing with transparency maps may require a little additional wizardry. The render-pass-based approach is much simpler. It is just a matter of replicating method a) except that the hold-out option is left on. This may correctly handle transparencies with no additional work.

It is best to resist the temptation to mask-out foreground objects from a beauty pass by using a scene matte. That method is more easier, but it handles transparencies incorrectly. Even if the scene only contains opaque objects, anti-aliased edges may have artifacts unless an advanced keying tool is used to apply the matte. Advanced keyers were designed to help solve the matting process for live-action footage, where it is not possible to "render" the background and foreground separately. In CGI we have the luxury of being able to isolate scene elements at the rendering stage. There is no excuse not to. It is futile to try to help compositors by rendering CG characters in front of "green screen" backgrounds.

Another related problem is that of producing a beauty pass with no background. Creating a separate render layer with the environment (or camera image plane) turned off works if the background is just a back plate. However, that solution is not acceptable if the environment has optical interactions with the contents of the scene, such as IBL. Objects in a scene with IBL are expected to have environment reflections, and environment lighting. An easier way to achieve the desired result is to render an occluded background image using one of the two hold-out methods described above, and to subtract that image

from the Master Beauty pass. This helps handle transparencies and anti-aliased edges correctly, and preserves optical interactions with the environment.

11.4. Dealing with Reflections and Refractions

Reflections and refractions are treated very similarly in rendering, and therefore can be composited using the same techniques. A simple way to isolate reflections and refraction is to render a beauty pass, and to set the minimum and maximum reflection and refraction levels in order to extract the desired trace recursion levels. **The dedicated *Reflection* and *Refraction* render pass types should be avoided** in most cases because they extract the reflection and refraction captured by the surface shader's BRDF, which is independent of PCMs. Using specially configured beauty passes to isolate reflections and refractions is better because they are sensitive to object and light visibility, as dictated by the PCM logic. One particular case where it is desirable to use the *Reflection* and *Refraction* pass types is with surface shaders that do not completely support the render pass framework, such as mia_material_x_passes (*c.f.* section 12).

The behavior of the Reflection and Refraction render pass presets can be changed by setting the *beautyPassTypeReflRefr* option variable. When this variable is set to 1, The *Reflection* preset creates a *Beauty* pass with the minimum reflection level set to 1, rather than a regular *Reflection* pass. This variable also has a similar effect on the *Refraction* preset. This option is off by default. Because it is an option variable, the setting will persist from one project to another (stored as a user preference). The following MEL command turns on the option:

```
optionVar -iv "beautyPassTypeReflRefrac" 1;
```

The following turns the option off:

```
optionVar -iv "beautyPassTypeReflRefrac" 0;
```

Reflections and refractions are composited back into a scene through simple addition (like pre-matted compositing). This process is a little bit different from pre-matted compositing of scene partitions because it does not require any actual pre-matting, since reflections and refractions are non-occluding phenomena. Therefore, it is possible for the compositor to apply geometric transformations to reflections and refractions without worrying about the occlusion hole problem. However, in some cases, it may be necessary to apply the coverage mask of the reflecting object to the transformed reflection in order to limit the reflection region to the domain of the reflective object.

11.5. Tone Mapping and Color Correction

Non-linear color transforms such as tone-mapping operators and color corrections are problematic for re-composing scenes from elementary render passes. This is because of the non-distributive nature of these transforms, as explained in section 5.4.3. The compositing logic discussed in this document assumes a linear color representation. Even when rendering to file formats that imply standard gamma correction, it is assumed that the compositing application internally converts the images to a linear color representation for the purposes of pixel arithmetic. Many non-linear color-manipulation facilities exist in mental ray. They usually take the form of utility shader nodes (for use in surface/volume shading networks), lens shaders, and output shaders. These functionalities are very convenient for obtaining finished results directly out of Maya. However, they should be avoided in a production pipeline that includes a compositing and/or color grading stage. Non-linear color transformations should be deferred down-stream. Otherwise, the compositing arithmetic is compromised.

If a particular mental ray lens shader or output shader provides a desirable color transform, it is still possible to use it as a post-compositing image processing step. One possible method is to render an empty scene with the image to be processed used as the camera background image plane. Using Maya and mental ray for image processing may

sound excessive; but it can save the trouble of reverse-engineering a desired tone-mapping function that is built-in to a proprietary shader.

Deferring the color transformation down-stream can have the undesirable effect of biasing creative compositing decisions because the images are not manipulated in their final appearance. This is a human visual feedback problem. To help solve this, several compositing applications allow the user to specify a tone map or color correction profile for on-screen display of images. Such a feature can be used to view intermediate images and compositing results alike in their tone-mapped or color-corrected form, while the underlying working data uses a linear color profile. In the Maya® Composite 2011 tool, this feature is known as a *display modifier*.

11.6. Using the Shadow Passes

Similar to transparencies, many traditional compositing workflows use single-channel shadow masks. These fail to capture shadow tinting, which may result from tinted shadow casters and colored lights. With mental ray for Maya, the shadow pass is an RGB image representing the difference in shading with and without shadows. This is equivalent to rendering a beauty pass with shadows enabled, a beauty pass with shadows disabled, and taking the difference between the two.

Shadows have a subtractive effect on the image. In order for the shadow image to be within standard viewing range, Maya stores the shadow images as positive, rather than negative values. Because of this, shadow compositing is performed by subtraction:

$$Beauty = BeautyWithoutShadow - Shadow$$

There is another related render pass type called the raw shadow pass. This is similar to the shadow pass except that it represents the difference in direct irradiance caused by shadows. This makes it possible to apply the shadow to objects that are re-colored or re-textured in compositing, since the raw shadow has the surface shading abstracted out. An important limitation is that only diffuse reflection can be accurately reproduced from direct irradiance information. The arithmetic for reconstructing a diffuse pass using a raw shadow is as follows:

$$Diffuse = DiffuseMaterialColor \cdot (DirectIrradianceWithoutShadow - RawShadow)$$

This equation gives the compositing artist the freedom to adjust shadows, illumination and material color/texture, and recompose a correct diffuse component.

An alternate approach is to reconstruct a shadow pass from a raw shadow and the non-diffuse components of the shadow. The first step is to isolate the non-diffuse components of shadows using existing render pass types:

$$NonDiffuseShadow = Shadow - DiffuseMaterialColor \cdot RawShadow$$

The non-diffuse shadow image represents the effect of shadow-casting on shading components other than diffuse reflections; for example, specular spots affected by shadows, shadows seen through reflections, translucence shadows, etc. The compositing equation for re-applying the full shadow with a decomposition that uses the raw shadow pass is:

$$Beauty = BeautyWithoutShadow - DiffuseMaterialColor \cdot RawShadow - NonDiffuseShadow$$

12. Working with the mia_material Shader

The mia_material shader is of great significance to many Maya users since it implements a very versatile physics based shading model. The multi-output variant of the shader,

mia_material_x, is very useful for extracting elements of the BRDF computation. In Maya 2009, mia_material_x was conveniently wrapped into a render-pass-enabled material called mia_material_x_passes that wires the outputs of mia_material_x into standard Maya render passes. The mia_material_x_passes material was greatly improved in Maya 2011 to perform better compositing in render passes and to support PCM based light contribution control.

12.1. Current limitations

As of Maya 2011, mia_material_x_passes is very close to having feature parity with Maya base shaders, with respect to the Maya render pass system. One of the main remaining limitations is that refraction and reflection rays cast by the material are bypassing the render pass framework. This means that render pass options for controlling min/max ray recursion levels are not respected.

Also, other useful surface shaders were wrapped into materials with the '_passes' suffix, indicating that the outputs of the original material shader are being redirected to standard render passes. A complete list of render passes written to by the *_passes family of shading nodes is given in the Maya User Guide, Rendering and Render Setup > Shading > Shading Nodes > mental ray for Maya shaders > mental ray for Maya nodes.

12.2. Extracting reflection and refraction render passes.

In section 11.4, it was stated that specially configured beauty passes are the preferable way of extracting reflections and refractions. That method does not work with mia_material_x_passes because of the limitation described above. This is also true for other non-render-pass-compliant shaders.

The alternative is to use the regular *Reflection* pass type to render reflections (*i.e.* the default behavior of the *Reflection* preset, with the *beautyPassTypeReflRefr* option turned off). This renders reflections correctly with mia_material_x_passes and the other *_passes materials, except that it outputs the reflections as produced for the Master Beauty pass. This means that the contents visible through the reflection are not subject to PCMs or other render pass settings. The same workflow also works for refractions, with the same limitations.

13. Working with the mental images Architectural Sun and Sky Shaders

There is a common complaint from users that use the physical sun and sky in conjunction with render passes: "Why are my render passes all blown-out?". This is not a bug with render passes. The problem is that the physical sun and sky is embodied by a small eco-system of shaders that includes a lens shader used for exposure control. This is the mia_exposure_simple shader, which is basically a tone-mapping operator. This tone-mapping is necessary for rendering to low dynamic range image formats. Because mia_exposure_simple is not render-pass-compliant, it only operates on the Master Beauty, and not on auxiliary render passes.

At first glance, this can be seen as a crippling limitation, but it actually isn't. The exposure shader applies a non-linear color transform, which invalidates simple linear compositing algebra that is used throughout this document, so that shader should be avoided in the first place. The most straight-forward solution is to work with non-tone-mapped or linearly

re-mapped images through the compositing pipeline, and to apply the final non-linear exposure control down-stream of compositing. The motivations and methods for this were explained in section 11.5.

14. Basic Compositing Techniques and Examples

This chapter exposes a few compositing techniques that show how to harness the power of render passes. The compositing screenshots are from Maya Composite 2011. These techniques use combinations of principles that were explained in earlier chapters, put into more holistic production-like contexts, although the scenes are simplistic for the purpose of demonstration. The scene files for the examples in this chapter are distributed with this document. Files are in Maya 2011 ASCII (.ma) format.

The techniques covered in this chapter suggest certain workflows, but there are usually many different alternatives to accomplish the same task. The intent of this chapter is not to cover all permutations and use cases exhaustively, but rather to provide a few initial sparks that computer graphics professionals can carry further.

14.1. Light Tuning

Balancing the lighting in a scene is a task that can easily be deferred to compositing, and is logical because it allows the artist to adjust the lighting interactively without having to re-render the scene. The scene file used for this example is LightTuningExample.ma. This scene has three sources of illumination: a lamp post, a mia_physicalsun, and a mia_physicalsky. The scene is rendered with final gathering enabled in order to get IBL illumination from the environment (sky). Render passes were set-up to isolate the different illumination components. The render passes are set to produce 3-component 32-bit floating-point images in OpenEXR file format. Colors are linearly encoded with Rec709 primaries. Here are the render results, with adequately adjusted gamma and exposure:

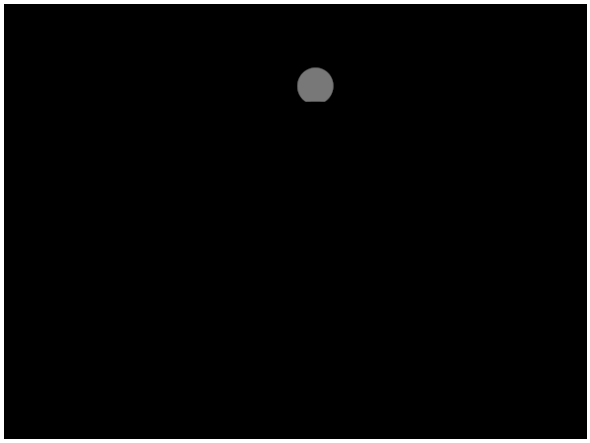


Figure 29 Incandescence

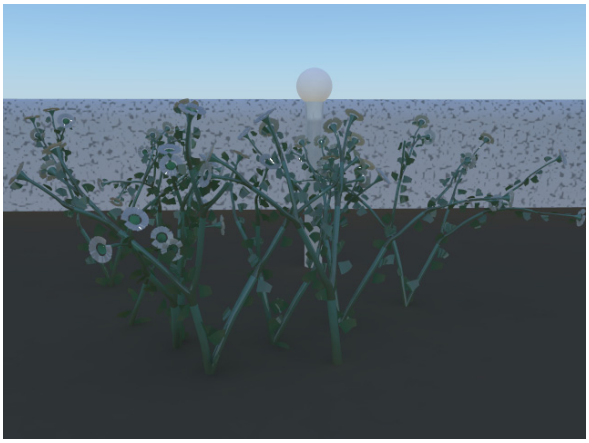


Figure 30 Environment

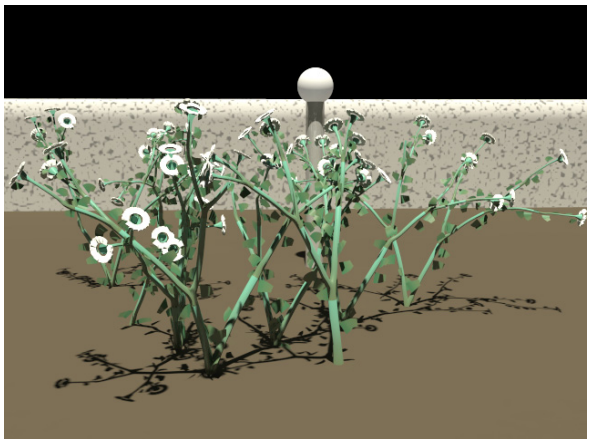


Figure 31 Sun Beauty

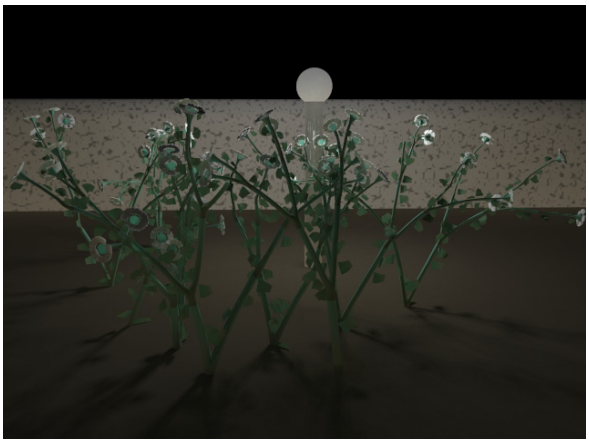


Figure 32 Lamp Beauty

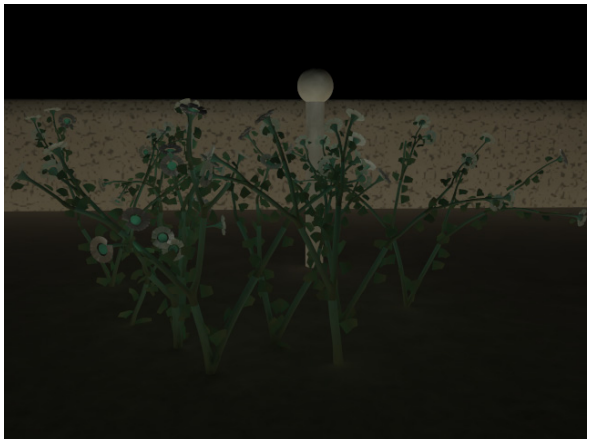


Figure 33 Indirect Illumination (sun and lamp)

Below is a description of the render passes, and explanations on how they were created. Additional details on the scene configuration can be found by consulting the scene file.

- Incandescence: A default incandescence pass, with no PCM.
- Environment: Beauty pass generated using PCM that excludes all lights in the scene except for a dummy point light that has an intensity of 0. The result is a beauty pass that contains direct light contributions from only the sky, and indirect light from all sources.

From an extra render layer with environment disabled:

- Sun Beauty: Beauty pass with a PCM that includes everything in the scene except for the lamp area light
- Lamp Beauty: Beauty pass with a PCM that includes everything in the scene except for the sun directional light
- Indirect Illumination: A default indirect illumination pass, with no PCM.

With minimal cleanup, recomposing the original image from its lighting components is as simple as adding them all together. The first step is to eliminate redundancies. The sun beauty and lamp beauty passes contain more than just the result of direct illumination from the sun and lamp. They contain the lamp incandescence, as well as the effect of indirect illumination. Therefore, an important preliminary step is to clean-up the beauty passes to isolate the direct illumination, which can be done by simple subtraction. The environment pass also needs to be cleaned-up by subtracting incandescence and indirect illumination from the lights. The following Composition does the trick:

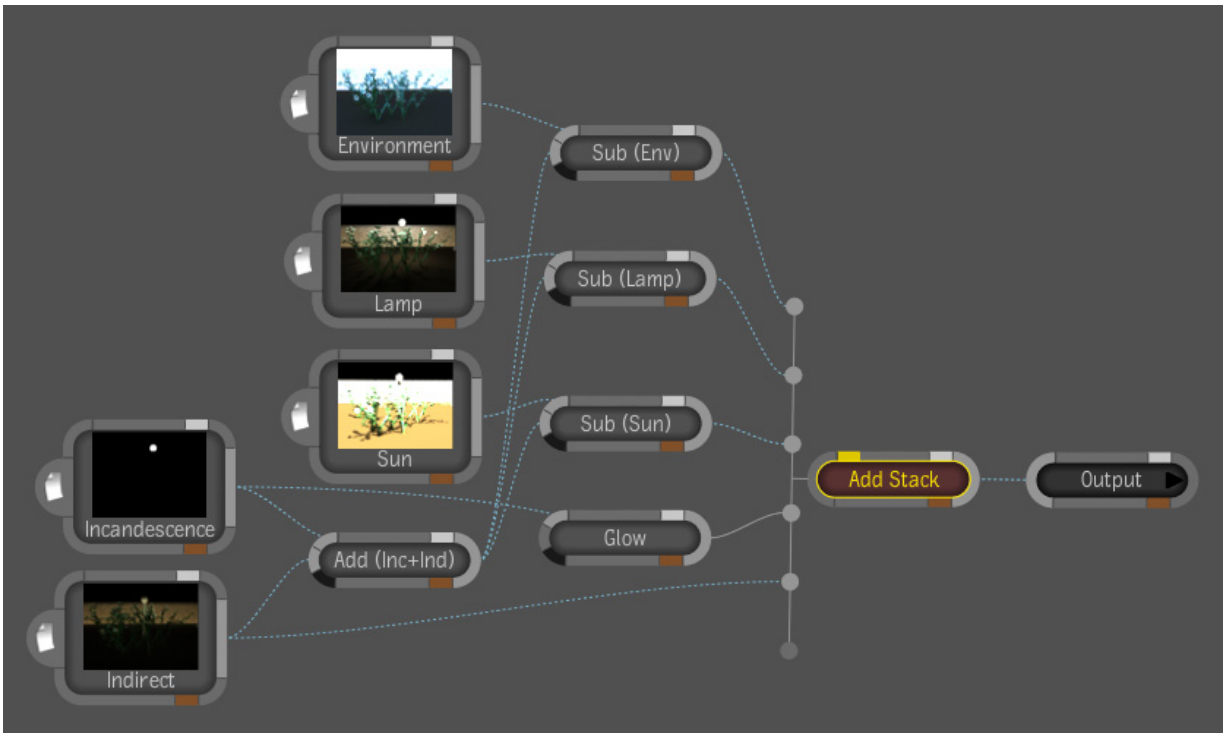


Figure 34 Composition for Illumination Recombination

The above compositing tree screenshot is from Maya Composite 2011, but it can be easily recreated in any other compositing software. The *Add Stack* node is an instance of the *Reaction* tool of the Maya Composite functionality. *Reaction* is a multi-layer compositing tool. In this case, the *Reaction* tool was simply configured to use the *Add* blending mode on all layers. The contribution amount of each layer can be adjusted by changing the *Multiplier* attribute of each layer in the *Reaction* tool. The glow tool is used to generate an artificial blooming effect around the lamp. The *Sub* and *Add* nodes are instances of the *MathOps* tool, set to *Subtract* or *Add*.

Just by tuning the multiplier values for each layer, it is possible to change the appearance of the scene to show what it would look like at dusk. To do so, the following factors were used on each layer:

- Environment: 0.065 (this controls the luminosity of the sky as well as indirect illumination from the sky
- Lamp: 1.0
- Sun: 0.0
- Glow+Incandescence: 1.0
- Indirect: 0.0 (most of the illumination in the indirect layer is from the sun)

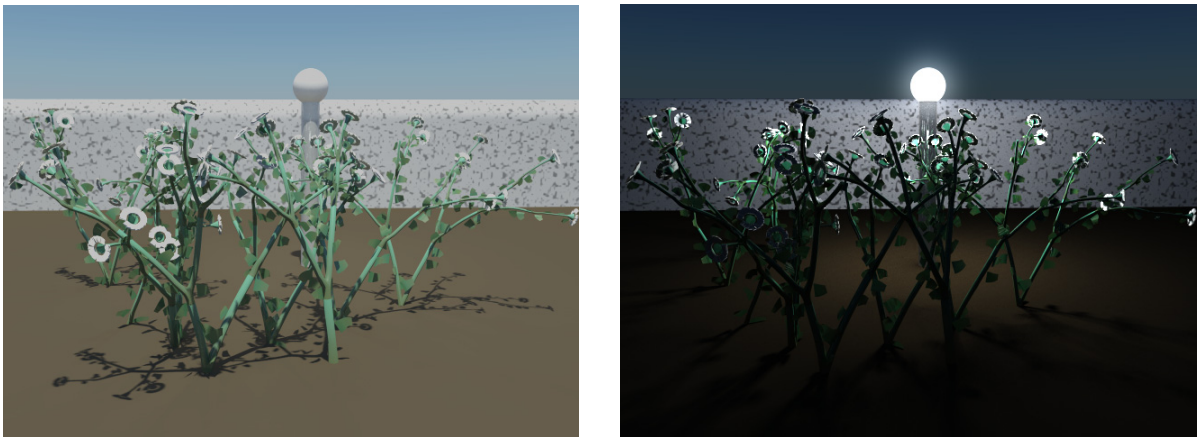


Figure 35 Original Render Result vs. Composited Result

The two images above are shown at different exposure levels because of the difference in illumination. The exposure of the composited result is 2.8 f-stops higher with no non-linear tone mapping, while the original render result has a default mia_exposure_simple lens shader applied to it.

By adding a color correction to the environment, it is possible to create the effect of an overcast sky. The idea is to decrease the color saturation of the sky, decrease the intensity of the sun, and increase the illumination from the sky, in order to simulate the dispersion of sun rays in the clouds.

This way of compositing an over cast day is not totally correct because although the sky is desaturated, as it should be, the indirect illumination from the sky IBL should not be as desaturated as the sky because the diffuse reflectivity of the objects in the scene is physically downstream of the clouds (as far as the physical light path is concerned). As a result, the scene may not be quite as colorful as it should be in reality. Instead, it is appropriate to separate the environment into two passes: the sky, and the scene lit by the sky. To do so, let us add a new render pass to our stack. The sky pass represents direct environment ray hits, and can be produced using method a) from section 11.3, but with hold-out enabled in order to obtain a pre-occluded sky for pre-matted compositing.

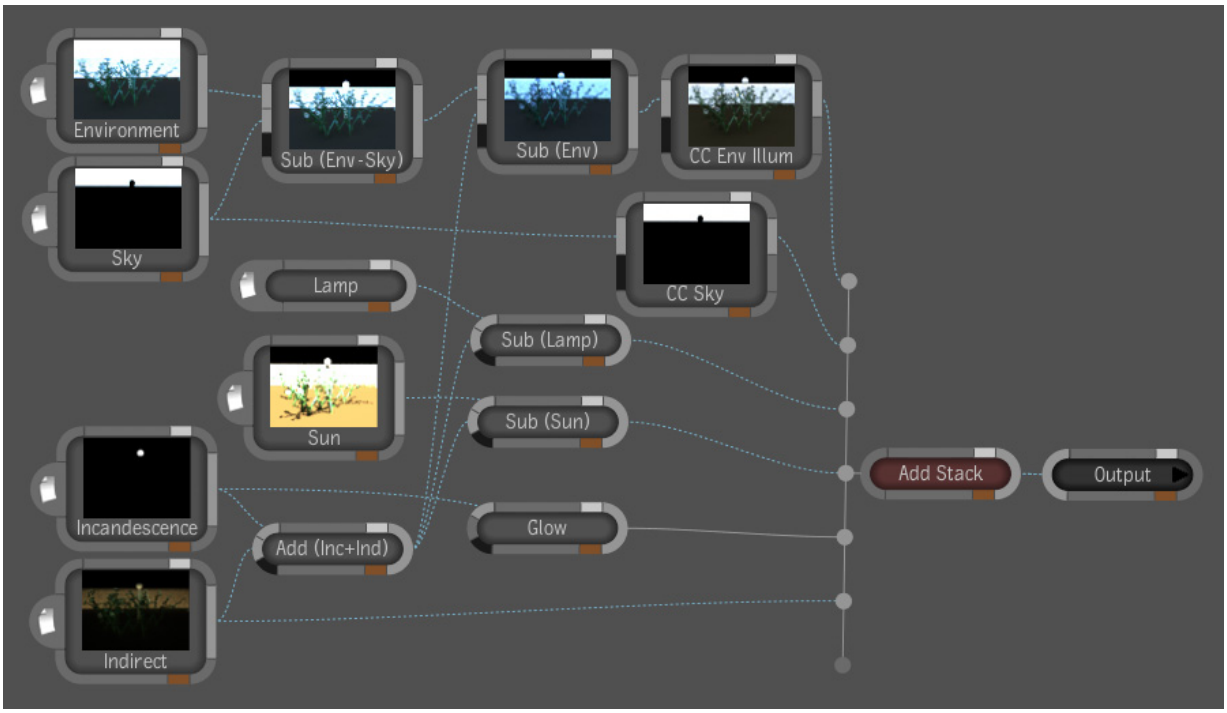


Figure 36 Composition for Overcast Day Effect

The saturation level set of the CC Sky node was set to 0.2. The *CCEnvIllum* node is a color correction node that adjusts the color balance of the environment illumination to neutralize the hue of the sky's illumination. The multiplication factors of the *Add Stack* node were set as follows:

- Environment: 1.0
- Sky: 1.0
- Lamp: 0.0 (lamp turned off)
- Sun: 0.25 (light cloud cover, sun still visible)
- Incandescence/Glow: 0.0 (lamp turned off)
- Indirect: 0.25

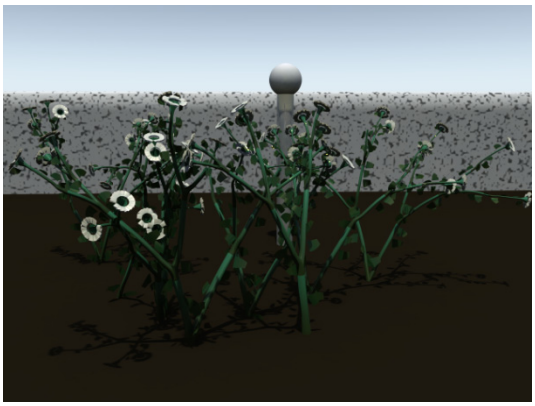


Figure 37 Gray Sky Effect

14.2. Shadow Tuning

The examples from the previous section included shadows, and they show that balancing the lights in the scene will balance the shadows proportionally, since the illumination component images were rendered with shadows. On the other hand, editing shadows without changing the illumination or vice versa is not usually a physically correct effect, but it is often performed as a compositing short-cut, or for artistic reasons. This can be achieved in compositing by applying a color-correction to a shadow pass or by blending-in shadow passes. This section examines the problems of dialing-in/out shadows, as well as the problem of modeling shadow caster opacity in compositing. The first examples in the section are based on sample scene ShadowTuningExample.ma. This is a very simple scene with two light sources and two shadow casters. For pedagogical reasons, the two lights have different hues and there are shadow intersections, and soft edges, which make it very clear how shadow formation behaves in the examples.

14.2.1. Dialing-in shadows globally

A simple and straight-forward way of controlling shadow intensity globally in the scene is to render the scene with and without shadows and to blend the two images in compositing. The blend factor will affect global shadow intensity. This is easy to setup

with two beauty render passes, one having shadows turned on, the other with shadows turned off.

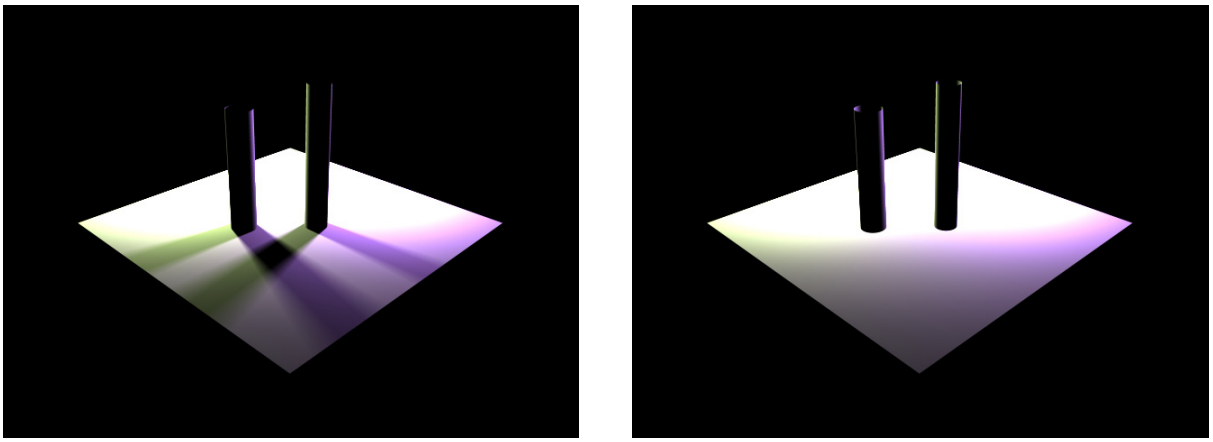


Figure 38 Beauty Passes with and without Shadows

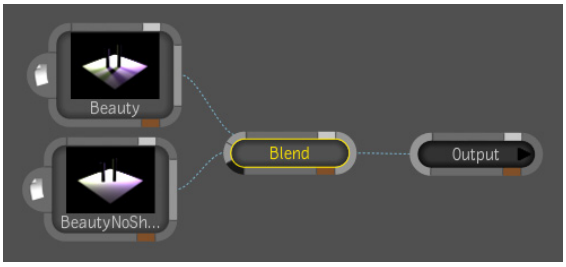


Figure 39 Shadow Blend-in Composition

There is, however, an interesting alternative: using a beauty pass and a shadow pass. This makes shadow editing more flexible while still remaining intuitive. The method consists of rendering a regular beauty pass without shadows enabled and a regular shadow pass. The shadow is applied by subtracting the shadow pass from the beauty pass. The advantage of this method is that interesting effects can be achieved by applying simple modifications to the shadow pass, such as a color correction node.

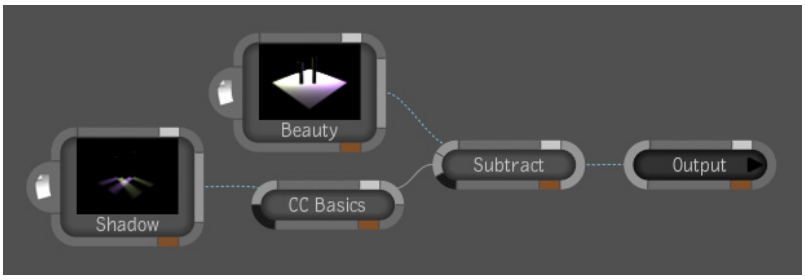
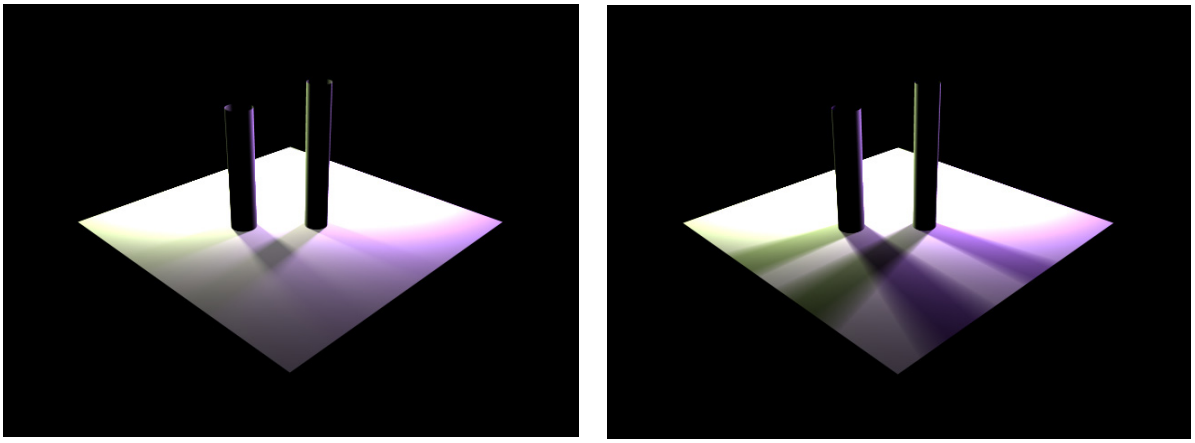


Figure 40 Color-corrected Shadow Composition

The CC Basics tool has a Gain parameter that can be used to adjust shadow intensity. Other creative effects can be achieved with this composition. For example, the tint of the shadows can be modified by adjusting the color balance in the CC node. Another interesting effect is to apply a gamma curve to the shadow pass. This affects shadow edge softness and fading.



Gamma = 0.57
Gain = 0.56

Gamma = 1.5
Gain = 0.75

Figure 41 Shadow Tuning with a Color Correction Node

14.2.2. Dialing-in Shadows per Light Source

The methods of 14.2.1 can be further ramified in order to break-down shadows on a per light basis, in order to control them independently. It makes sense to combine this approach with an illumination composition in order to perform light tuning and shadow tuning simultaneously.

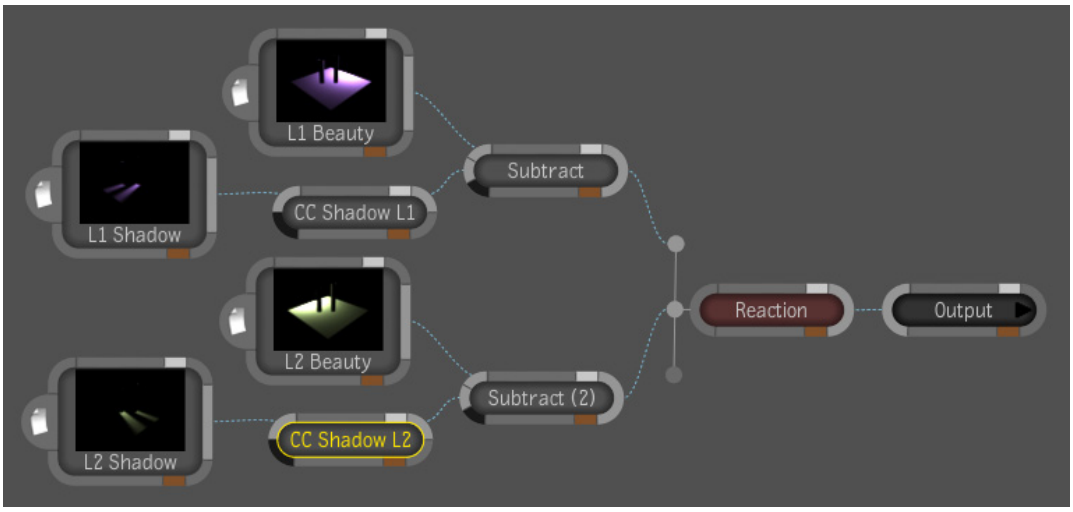


Figure 42 Light and Shadow Tuning Composition

As before, all layers of the reaction node use planes with the *Add* blend mode.

14.2.3. Modifying the Shadow Opacities of Shadow Casters

The shadow opacity of a surface is defined as the fraction of a shadow ray that is occluded by a surface. In most mental ray materials, this is usually related to the “transparency color” attribute of the surface shader. It may, however, be controlled by a separate parameter to distinguish between eye-ray transparency and shadow ray transparency; or, it can be controlled completely independently by a shadow shader. Controlling shadow opacity can also be done in compositing.

The first step is to render the shadows that are projected by each shadow-casting object of interest. The naïve way of isolating these shadows is to use PCMs to remove objects that we do not want to get shadows from. This method breaks down in the presence of overlapping shadows, or when an object that we don’t want shadows from is a shadow receiver. Take the following example of three superposed planes.

Figure 43 Cascading Shadows

A simple way to isolate the shadows cast by the top plane is to create an additional render layer and to use layer overrides to prevent all other objects from casting shadows. This behavior is controlled by the *Casts Shadows* attribute on the shape node. The scene file ShadowTuningExample2.ma is set-up to produce the following passes.

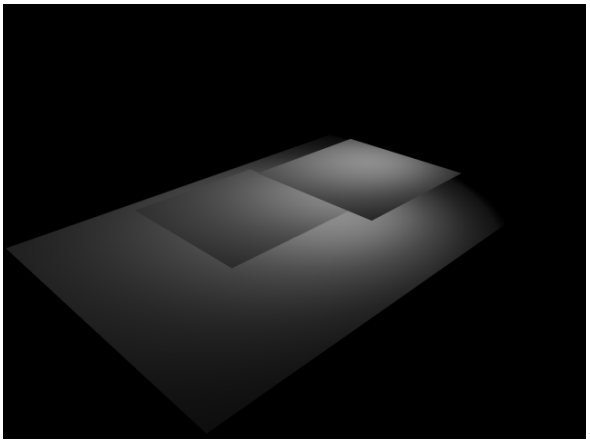


Figure 44 Beauty without Shadows

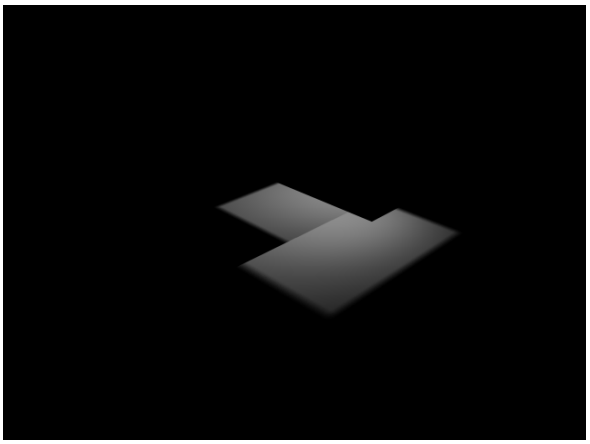


Figure 45 Shadows Cast by Top Plane

Figure 46 Shadow Cast by Middle Plane

A noteworthy detail is that there is an area of overlap in the 2 shadows. This will make it trickier to combine the two shadows in compositing. The following compositing graph performs the proper accumulation of the two modified shadows:

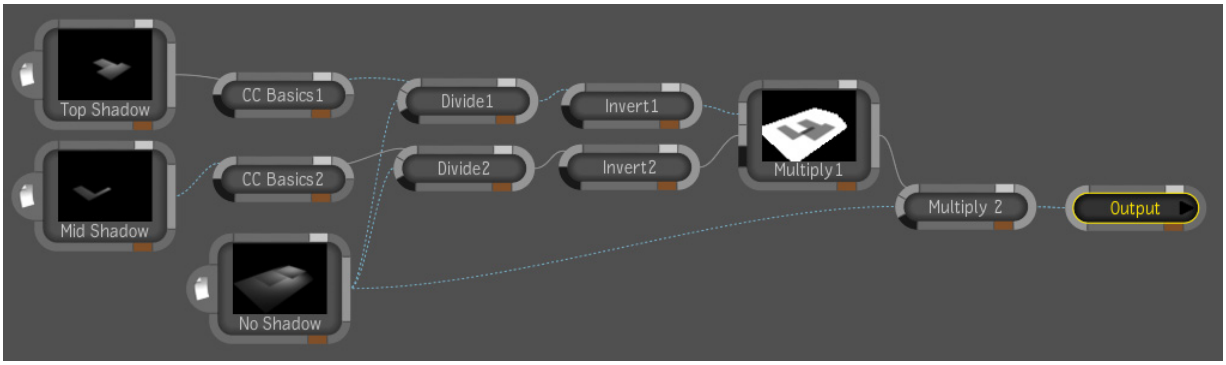


Figure 47 Composition for Combining Shadows

The two *CC Basics* nodes serve as shadow modifiers. The rest of the graph is for recombining the shadows. The *Divide* nodes (actually *MathOps* nodes that were renamed) are used to compute the fraction of scene illumination blocked by the individual shadows. The *Invert* nodes compute the fraction of illumination not blocked by the shadows. The *Multiply1* node combines the shadow transmittance fractions, which are then used to modulate the un-shadowed image with the *Multiply2* node.

The following image was produced by setting the Gain to 0.5 on both CCBasics nodes, which corresponds to reducing the shadow opacity of both planes by 50%.

The example above is somewhat simplified because it operates under the assumption that the entirety of the shading in the un-shadowed beauty pass is a result of direct illumination. A more general solution would involve factoring-out components that are not results of direct illumination, and processing each reflection and refraction trace level independently.

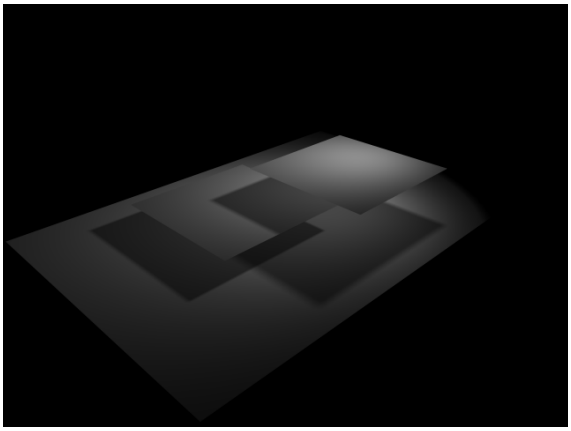


Figure 48 Attenuated Shadow Opacity

14.2.4. Re-projecting Shadows

A very important application for shadow passes is the application of rendered shadows onto live-action footage or CG content that was rendered separately. The usual method for creating the shadow passes involves setting up stand-in surfaces that act as shadow receptor proxies for the geometry in the scene that receivesreceive the composited shadows. With live action footage, this task is often accomplished using a technique called match-moving.

This compositing technique can be illustrated using the ShadowTuningExample2.ma scene, also referred to in the previous section. An additional beauty pass was added to the masterLayer with a pass contribution map that captures the top and middle planes. This new pass is generated with an alpha channel (coverage mask), which is necessary later for use as a compositing matte. This pass has the content that we want to composite onto an alternate background.

The bottom plane is the proxy shadow receiver. The shadows are captured using two direct irradiance passes, one with shadows enabled, the other with shadows disabled. The direct irradiance passes are rendered un-occluded by using a PCM to exclude the foreground objects, and the hold-out option is disabled.

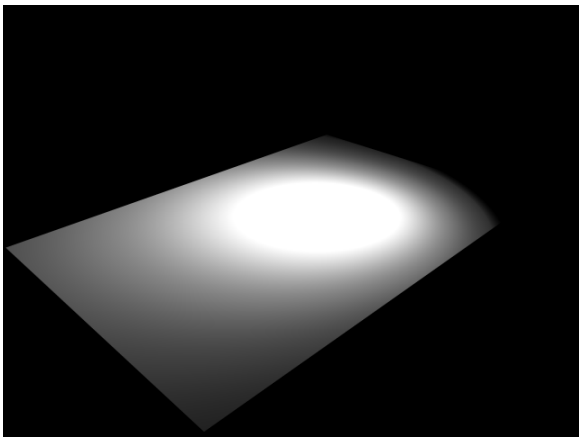


Figure 49 Direct Irradiance Without Shadows

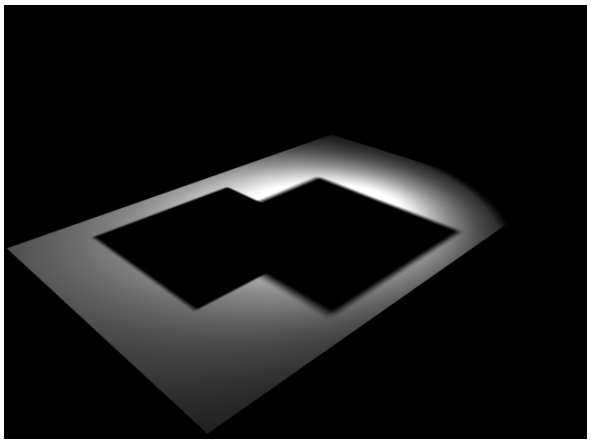


Figure 50 Direct Irradiance With Shadows

Using these two images, computing the shadow transmittance fraction is a trivial division operation. Just as in 14.2.3, applying the shadow is a matter of multiplying the shadow receiver image by the shadow transmittance fraction.

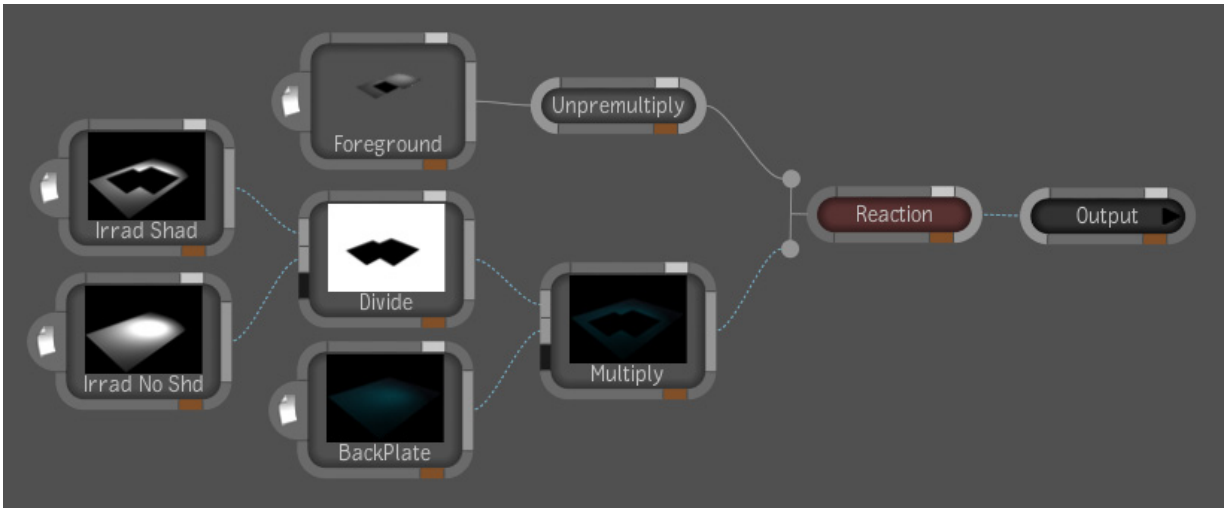


Figure 51 Composition for Shadow Re-Projection

The back plate is simply a new ground plane with lighting conditions identical to the original scene. The *Reaction* node uses a plane for the foreground layer, which is composited using the *Normal* blend mode (which uses the alpha channel as a matte). It is important not to forget the *Unpremultiply* node, which is necessary to prevent dark artifacts along the anti-aliased edges of the foreground layer.

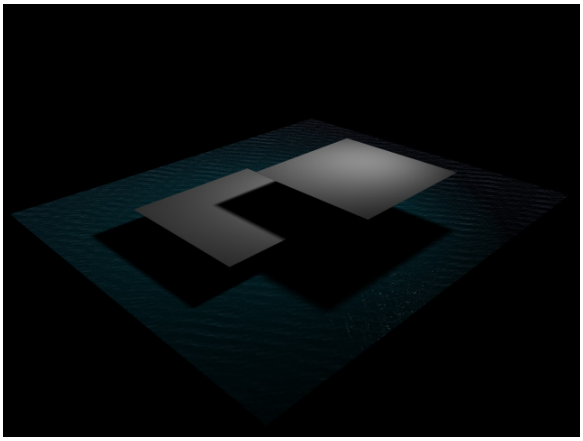


Figure 52 Shadow Re-Projection Result

14.3. Managing Lighting with Partitioned Scenes.

The shadow and light tuning methods shown in the previous section can get very complex very easily when combined with scene partitioning. The difficulty added by scene partitions is that of combinatory coverage. When decomposing a scene into geometry partitions and light partition, it is important to make sure that no light emitter/receiver pair is left-out or duplicated.

14.3.1. The Combination Matrix

The principles of scene partitioning are not exactly rocket science, so it may be tempting to "wing it", which is a good way to get into trouble. Constructing a combination matrix is a simple —and not very time consuming— method for obtaining complete and non-redundant coverage of all direct illumination interactions in a scene. The idea is to construct a table where there is one row for each partition of renderable geometries in the scene (surfaces and volumes), and one column for each light partition. The matrix cells must be filled-in with the render pass identifiers (numbers, names, colors, whatever). The idea is to fill-in the matrix. Ideally, the matrix should be full, with no overlaps.

Here is an example of a matrix for a scene with four scene partitions, corresponding to 4 non-intersecting PCMs, and the union of which covers the entire scene. There are also four illuminant groups.

Combination Matrix Example

	Sun	Sky	Key Light	Ambient Light
Ground	A	A	B	B
Set	C	C	D	D
Character	E	F	G	H
Props	C	C	I	J

Obviously, we could have just assigned a different render pass to each cell in the matrix, but that can lead to a compositing set-up that is heavier than necessary. In the case above, we decided to group together elements that are to be re-lit together. For example, pass C contains both the set and the props as lit by the sun and the sky. The decisions as to which partitions are to be grouped together don't necessarily need to be made at rendering time. Rendering a finer grained set of passes (like one for each cell) leads to more flexibility in compositing, because nothing prevents passes from being added and subtracted from each-other to re-combine partitions in an intermediate step. The recombined (intermediate) passes can then be used to fill-in the matrix.

It is not always necessary to fill the matrix completely when certain objects are known not to be lit by certain lights (*e.g.* when using light linking).

14.3.2. Reflections and Refractions

Scene partitioning has important implications for reflections and refractions because scenes from different partitions can cast reflections and refractions of each other. This is resolved using either of the methods proposed in section 11.2.1. These solutions make optical interaction problems easier to resolve when renders are partitioned using render passes (as opposed to render layers).

14.3.3. Shadows

Shadows add an additional level of difficulty because the combination space that needs to be covered is three-dimensional (light source, shadow caster, and shadow receiver). A matrix approach can still be used, but may be excessive. There typically aren't as many shadows in the scene as there are possible light/caster/receiver combinations, so it is usually worth taking the time to manually identify the shadows that need to be isolated by manually inspecting the Master Beauty pass.

14.3.4. Indirect Illumination

Indirect illumination (i.e. final gathering or global illumination) is a tricky case because it is computed globally, and not on a per-render pass basis. The example in section 14.1 showed that the indirect illumination from various light sources was not dissociable at the compositing stage within a given render layer. The trick to overcome this while resolving the reflection/refraction visibility issue (inherent to partitioning) is to leverage render layers and passes simultaneously. Render layer memberships can be used to control light source contributions, while PCMs are used to control the contributions of scene's geometric entities. For example, to render pass "C" from the above matrix, one would use a render layer that contains the entire scene except for the key light and the ambient light, and a PCM that contains the Set and Prop objects. The result would be a beauty pass that combines both the direct and indirect illumination from just the sun and the sky on the set and prop objects, while still being able apply the methods of section 11.2.1.

14.4. Tuning Reflections and Refractions

Using the minimum and maximum reflection and refraction level parameters, it becomes possible to decompose renders into specific trace levels that can be recomposed by addition. This decomposition method poses a combinatorial problem very similar to that of decomposing lighting with scene partitions (*c.f.* section 14.3). In this case, what is sought is a complete and non-redundant coverage of all reflection and refraction level combinations. This problem can be modeled very simply, using a similar matrix method to represent reflection level versus refraction level. For example, suppose we want to be able to tune based on four passes: direct eye rays, first level reflection, first level refractions, and everything else. The question is: what exactly is "everything else", and how is the image of that obtained? The matrix representation helps visualize the problem.

Trace-Level Coverage Matrix: Example of incomplete coverage

		Reflection Level		
		0	1	2…N
Refraction Level	0	A	B	
	1	C		
	2…N			

Unfortunately, because of the nature of the parameterization (min/max), a render pass can only cover a rectangular region of the matrix. In this example, the remaining areas can be

covered using three render passes, which can be added together later to form an intermediate “everything else” pass.

Completed Trace-Level Coverage Matrix

		Reflection Level		
		0	1	2…N
Refraction Level	0	A	B	E
	1	C	F	F
	2…N	D	F	F

Passes *D*, *E* and *F* simply need to be added together to form the remainder image. When using this systematic approach correctly, no trace recursion levels are left-out or duplicated in the final composition.

14.5. Deferred Motion Blur and Depth of Field

Motion blur and depth of field are two very important effects that replicate physical phenomena that occur in cameras. These effects are very important in cinematography in general, and not just in rendering. Most commercial renderers, including mental ray, are capable of rendering these effects in a physically-based fashion, but the computational cost is often very high. An alternative is to use image-based algorithms for approximating these effects with compositing software. The image-based approach has the advantages of allowing the effect to be adjusted interactively without having to re-render the 3D scene, and significantly cutting-down on rendering time. However, the results are usually not quite physically correct; although, in many cases, they are a good enough approximation.

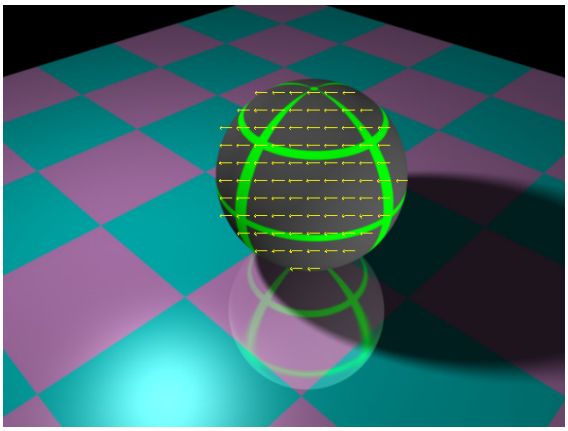


Figure 54 Beauty with Pass Motion Vectors Overlaid

14.5.1. Image-Based Motion-Blur

A motion blur effect can be added in compositing by simply rendering a motion vector pass and feeding it into a blur tool. In the Maya Composite 2011 feature, the blur tool has a forward vector input slot that was specifically designed to generate motion blur from motion vectors. Sample scene MotionExample.ma illustrates how to set-up an animated scene to produce a 2D motion vector pass that can be used for this purpose. It is really important to use the 2D Motion vector pass type, and not the X and Y components of a 3D motion vector pass. These are very different. The 2D motion vectors are in screen-space coordinates (perspective projection). The example scene was also specially designed to demonstrate several shortcomings of this method.

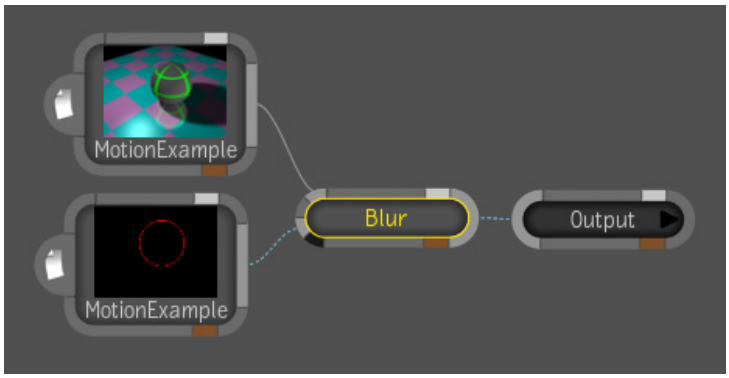


Figure 53 Motion-Blur Composition

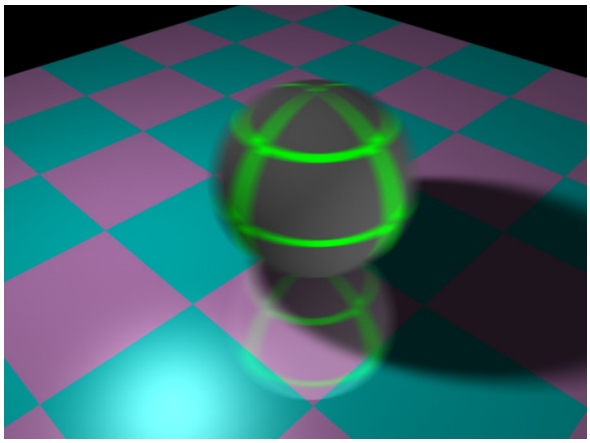


Figure 55 Rendered Motion-Blur

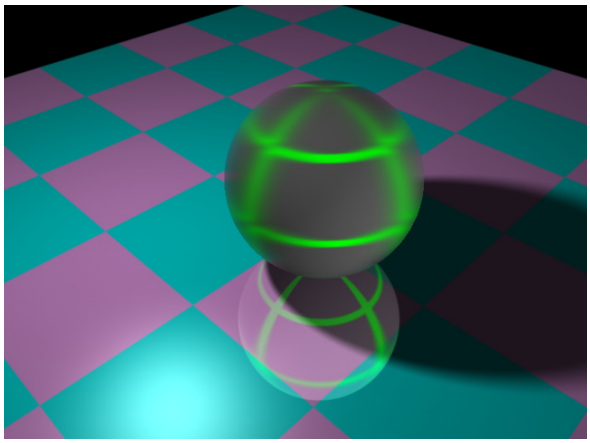


Figure 56 Vector-Based 2D Motion-Blur

There are several problems with vector-based motion-blur generation. The most obvious one is the poor handling of discontinuities in the vector field, leading to the improperly blurred edges of the sphere. The other predominant problem is that motion vectors only represent the motion for primary ray hits. Therefore reflections, refractions and shadows are not blurred. A more subtle discrepancy is the blur filter profile. The blur tool in the Maya Composite 2011 feature applies a Gaussian blur, which does not match the camera shutter profile used by mental ray, which explains why the blurred grid texture on the ball does not look exactly the same in the two images.

The Maya Composite 2011 system has a built-in mechanism for resolving the edge issue by using the *Extend Alpha* option on the *Blur* tool. This option grows the motion vector field to cover the area occupied by the moving object's edges. Good results can be obtained with this method by partitioning the scene into object groups with continuous motion vector fields.

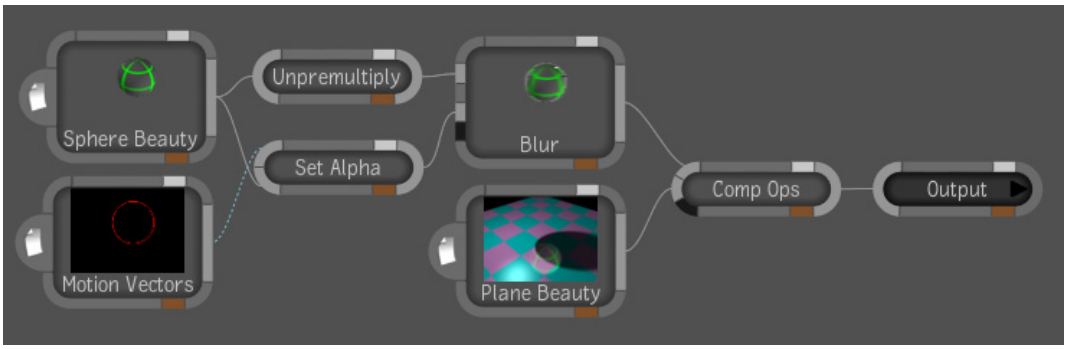


Figure 57 Motion-Blur Composition using Extend Alpha

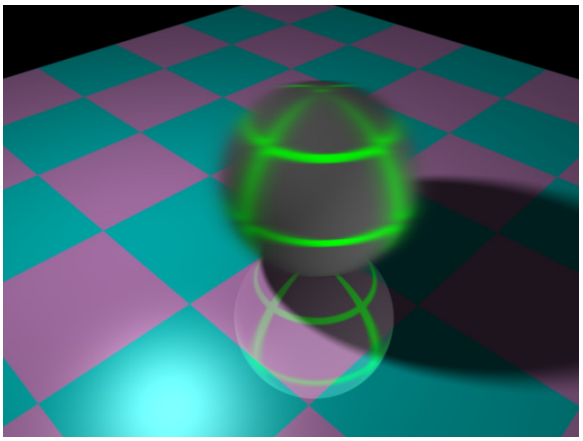


Figure 58 Composited Motion-Blur using Extend Alpha

This is a good improvement over the result of Figure 56 but the motion-blur effect is still not complete. One way of faking motion-blur on shadows, reflections and refractions is to decompose the image into components that can be blurred individually. With this method, the blurring of secondary optical effects does not use motion vectors and it requires quite a bit of manual tweaking.

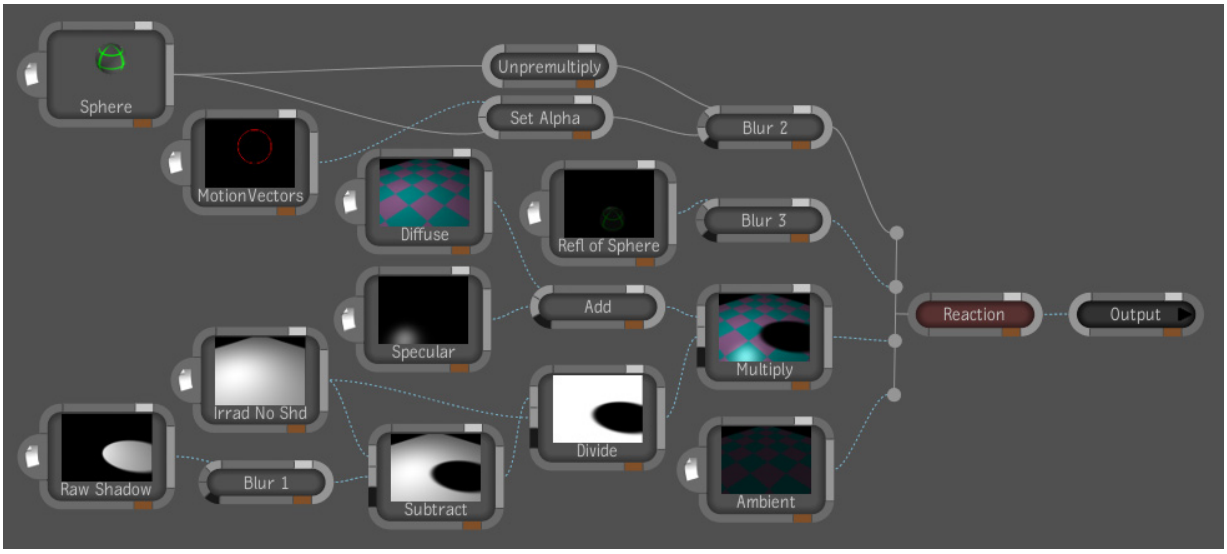


Figure 59 Composition for Faking Motion-Blur by Component

In the above composition, the shading of the plane had to be decomposed into its ambient, diffuse and specular components. This is simply because of the presence of an ambient light in the scene, and the fact that the re-application of the blurred shadow must not affect the ambient shading component. The *Reaction* node composites the three bottom layers additively (pre-matted compositing), while the sphere layer is composited with a matte in order to get correct occlusion with the blurred edges. The three *Blur* nodes in the composition give separate control over the directional blurs that are applied to the shadow, reflection, and direct image of the sphere. Although this method yields a more complete effect, it is still less accurate than what can be achieved by a renderer, in part because the method uses uniform blurs on the secondary optical interactions. Real motion-blur is rarely perfectly uniform because of perspective projections, rotational

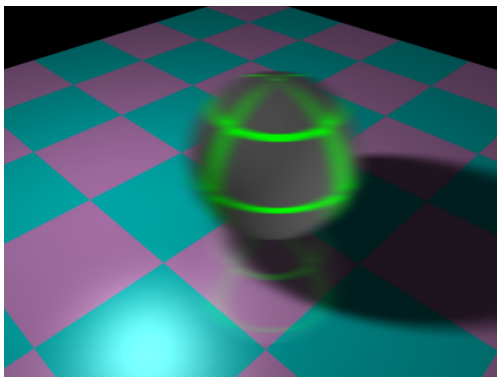


Figure 60 Result of Faking Motion-Blur on the Reflection and Shadow

motion, local deformations, etc. For simple cases, however, results are often acceptable.

14.5.2. Image-Based Depth of Field

Adding a depth of field effect in compositing is very similar to adding motion-blur. The difference is that a Camera Depth render pass is used to modulate the blur. One of the difficulties, however, is converting the depth pass into a map that is adequate for blur modulation. Raw physical depth does not produce a compelling result. One approach is to take 1/depth, as in the following example.

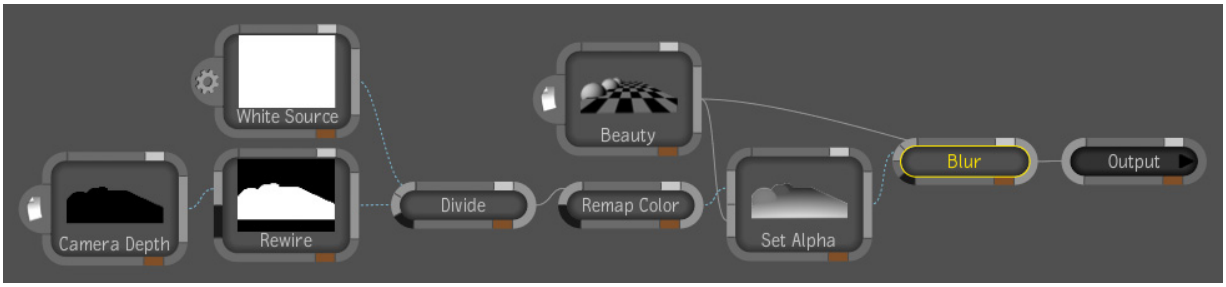


Figure 61 Composition for Depth-Based Blur Modulation

The *Rewire* node is necessary because the Maya Composite 2011 feature interprets the Z-channel as alpha, and this node is set-up to rewire alpha into the color channels. The *Remap Color* node only performs a input range adjustment to bring the inverse depth values into the [0,1] range. The blur node uses the Extend Alpha feature to handle edges.

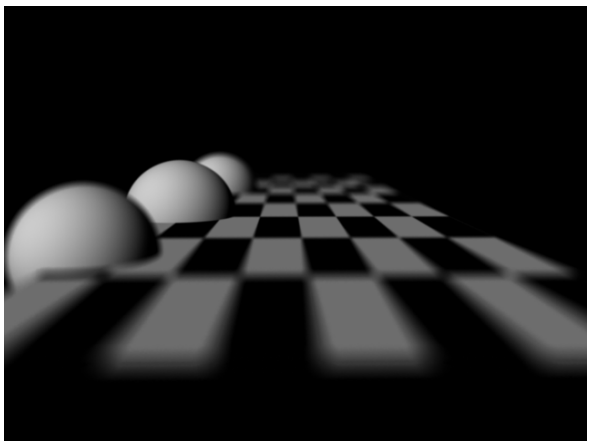


Figure 62 Depth of Field Rendered with mental ray

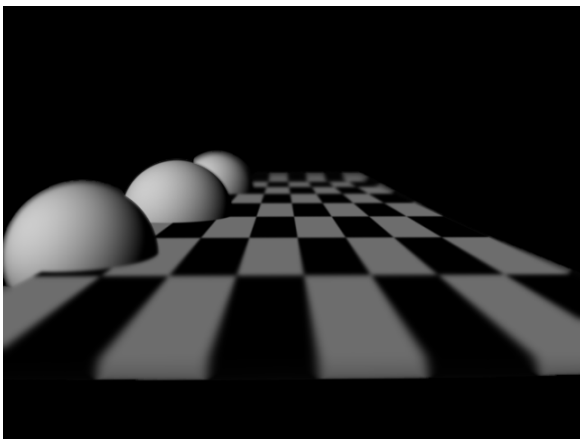


Figure 63 Composited Depth of Field Effect

This use case has much of the same pitfalls as vector-based 2D motion-blur. Many of the same solutions can be used to help resolve the problem. Artifacts around edges can be partially resolved by partitioning the scene and blurring by partition. As far as reflections and refractions are concerned, faking it (snake-oil-style) may be attempted by manually blurring the reflection and refraction component images, but getting near-correct results is a lot trickier than with motion-blur. For shadows, nothing special needs to be done, as long as the depth of field effect is applied to the shadow receiving objects.

Figure 63 also shows that edges at high viewing angles appear to be curled (front and back edges of the checkered plane). This is simply a side effect of isotropic blurring, combined with the fact that the Extend Alpha feature does not do any extrapolation. These limitations prevent simple blur modulation from accurately representing the depth of field phenomenon. Nonetheless, in many less extreme situations, this method is adequate for rendering a convincing effect at a fraction of the cost in rendering time (because rendering DOF requires high sampling levels to get smooth results).

14.6. The De-comp Re-comp Workflow

Planning a complete re-composition from a vast set of elementary passes can be extremely challenging, especially when combining multiple types of decompositions (scene partitioning, illumination decomposition, BRDF decomposition, shadow decomposition, reflection and refraction-level decomposition, *etc.*) If the render passes and the compositing tree are not configured meticulously, there can be elements that are either missing or applied multiple times in the final composite.

The de-comp re-comp workflow requires less planning. The compositing process is ad-hoc in the sense that the artist decides on-the-fly which components need to be factored-out and retouched independently.

The basic idea is to start with a global beauty pass as an initial canvas. Then, any component that needs to be re-touched is subtracted-out, modified independently, and added back in to the final result. This method is not fool-proof, however. Incoherent results may occur when overlapping components are retouched independently. For example, if a given object's diffuse color is changed, and later, a shadow that is cast onto the same object needs to be edited, then there is a problem. The process of subtracting-out the shadow is difficult because the rendered shadow pass does not respect the new diffuse color of the object.

It is also possible to execute a hybrid workflow, where the de-comp re-comp method is used on individual branches of the global compositing tree, *e.g.*, scene partitions.

15. The Shader SDK

The render pass system in mental ray for Maya is a powerful tool, but one of its main caveats is that it does not work completely with regular mental ray shaders. For some features to work correctly, it is necessary to use shaders that were specifically designed to support render passes. In order to help shader developers produce mental ray shaders that are render pass compliant, there is a shader SDK. Shader writing is beyond the scope of this document. Reference documentation on the SDK is embedded in the SDK header files. The SDK files are located in:

<Maya Install Directory>/devkit/adskShaderSDK

Unlike the mental ray shader API, this SDK is not in standard C. It is in C++ and uses many C++-specific language constructs, which may be foreign to some mental ray shader writers. The implementation patterns used in the SDK are also somewhat non-traditional for shader writing. To help users understand the implementation patterns, two sample shader implementations are provided with the SDK. These examples show two different ways of implementing a simple Phong surface shader.

As of Maya 2011, there are no published examples for writing other types of shaders that also need to be render-pass compliant: light shaders, volume shaders, shadow shaders, shading graph utility shaders, lens shaders and output shaders. Additional assistance can be obtained on-demand through the Autodesk Developer Network.

Appendix – Flame Compositions

The following screenshots illustrate how to build compositions similar to those in section 14 using Autodesk Flame 2011 software. The compositing results may be slightly different since Flame and the Maya Composite feature use different implementations and algorithms. In particular, the blur tools are very different.

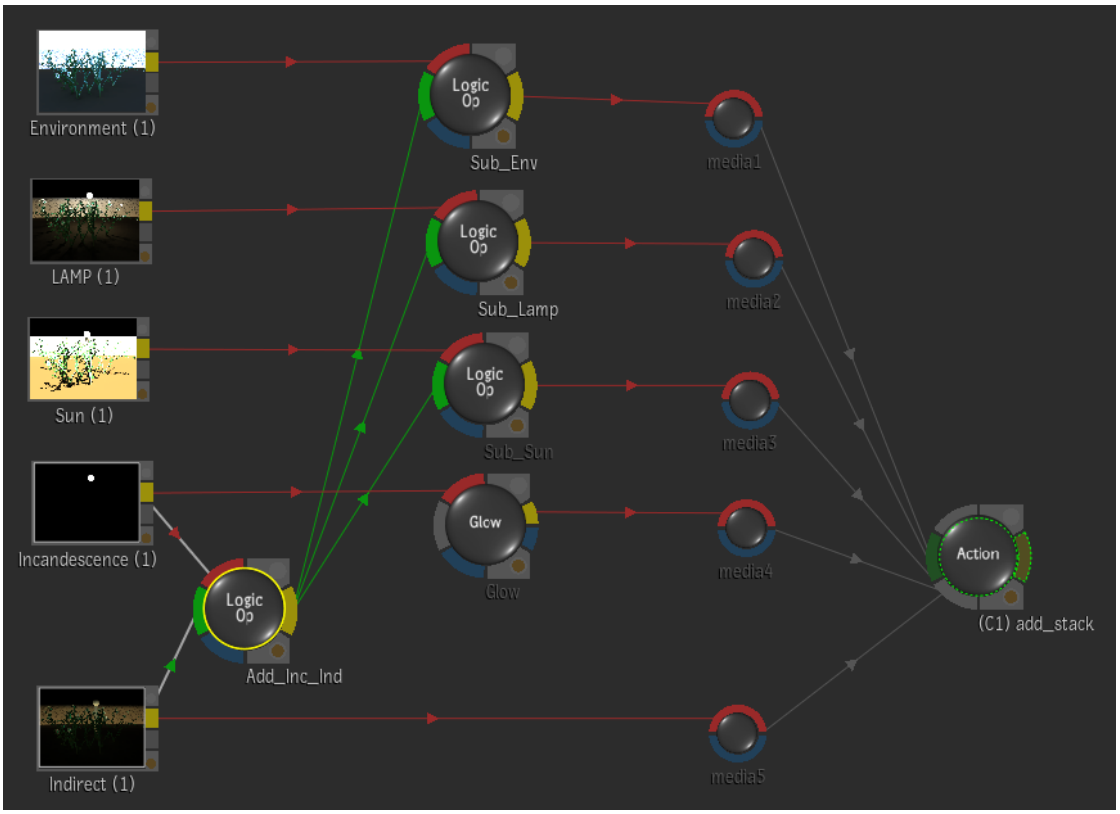


Figure 64 Illumination Recombination

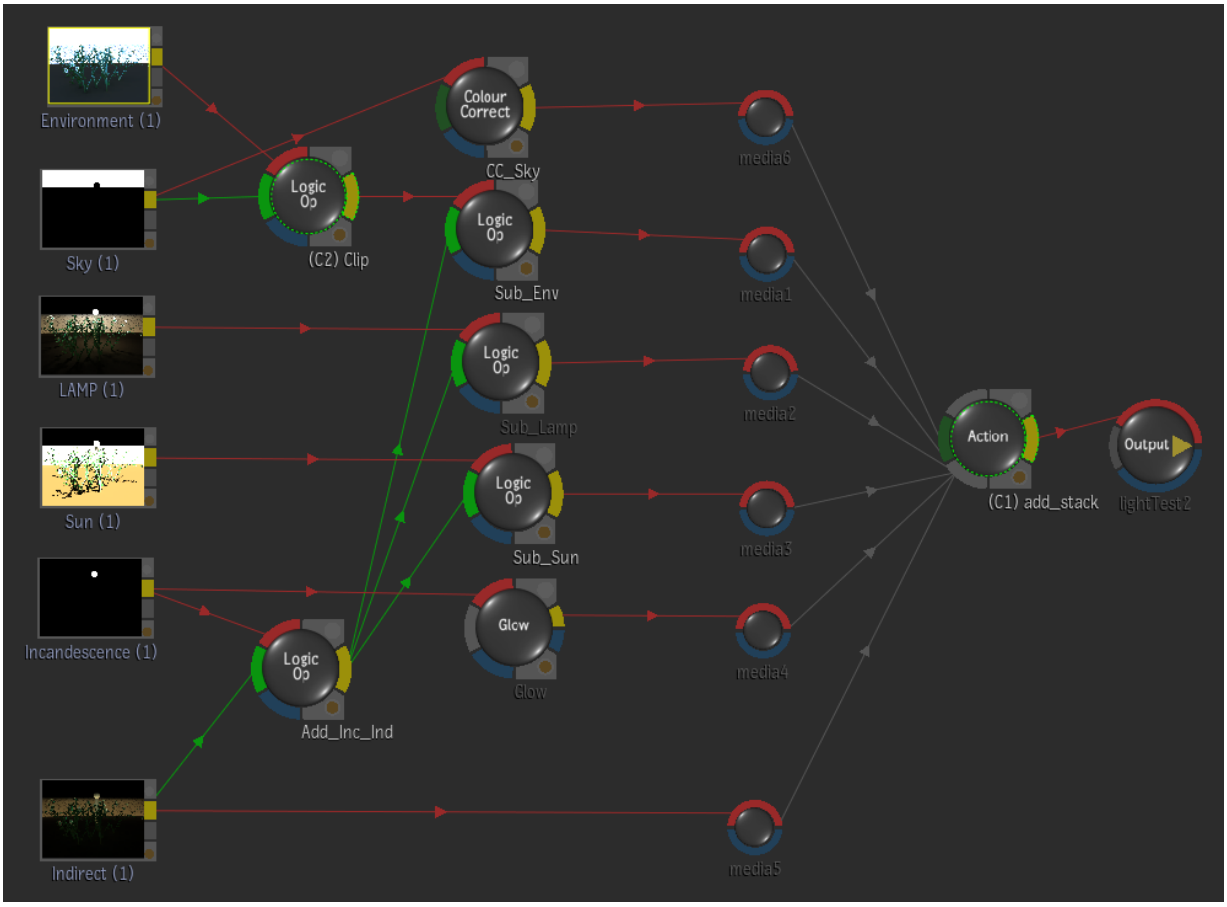


Figure 65 Overcast Day Effect

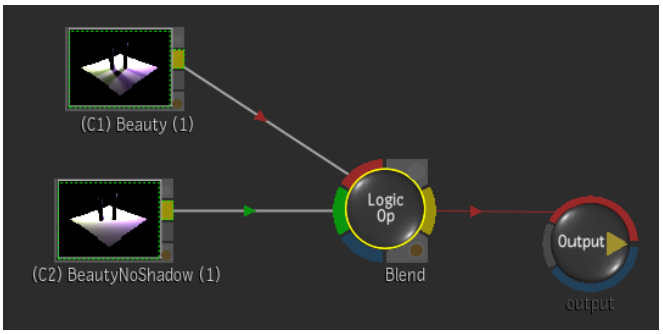


Figure 66 Shadow Blend-In

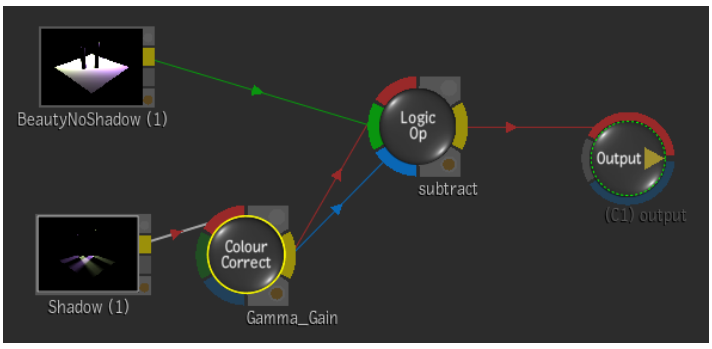


Figure 67 Color-Corrected Shadow

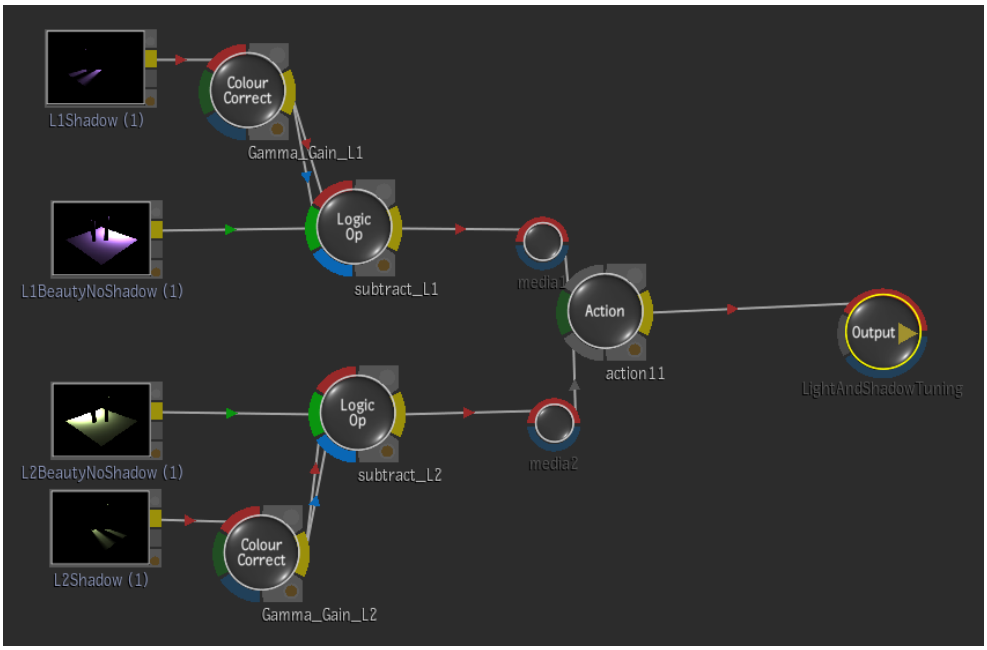


Figure 68 Light and Shadow Tuning

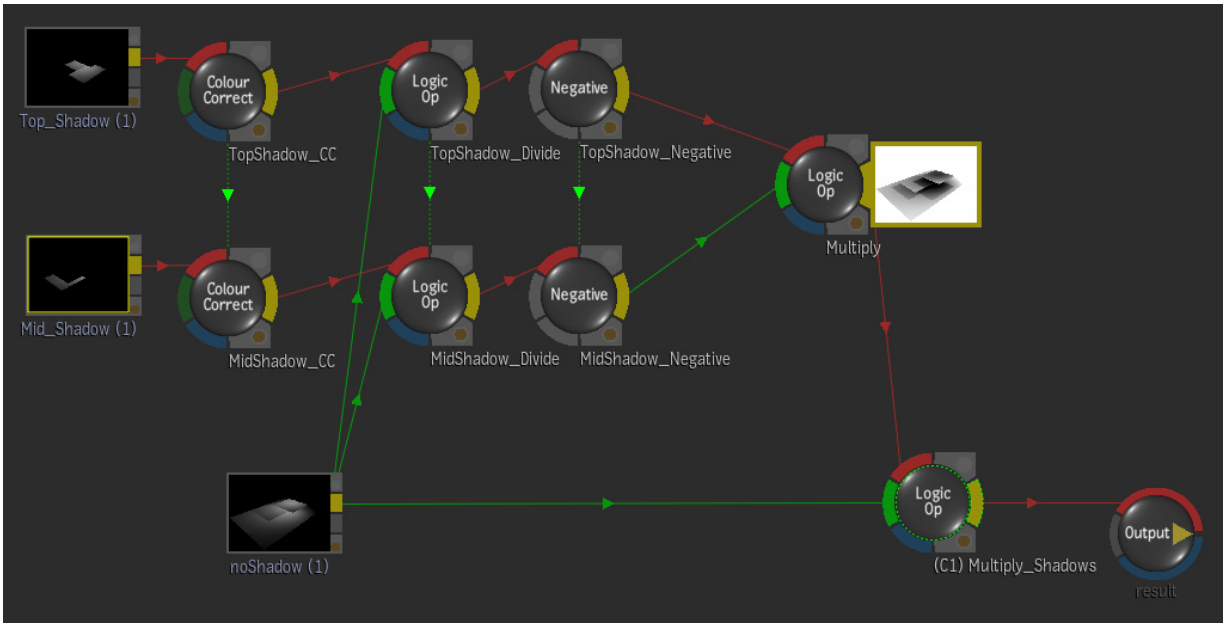


Figure 69 Combining Shadows

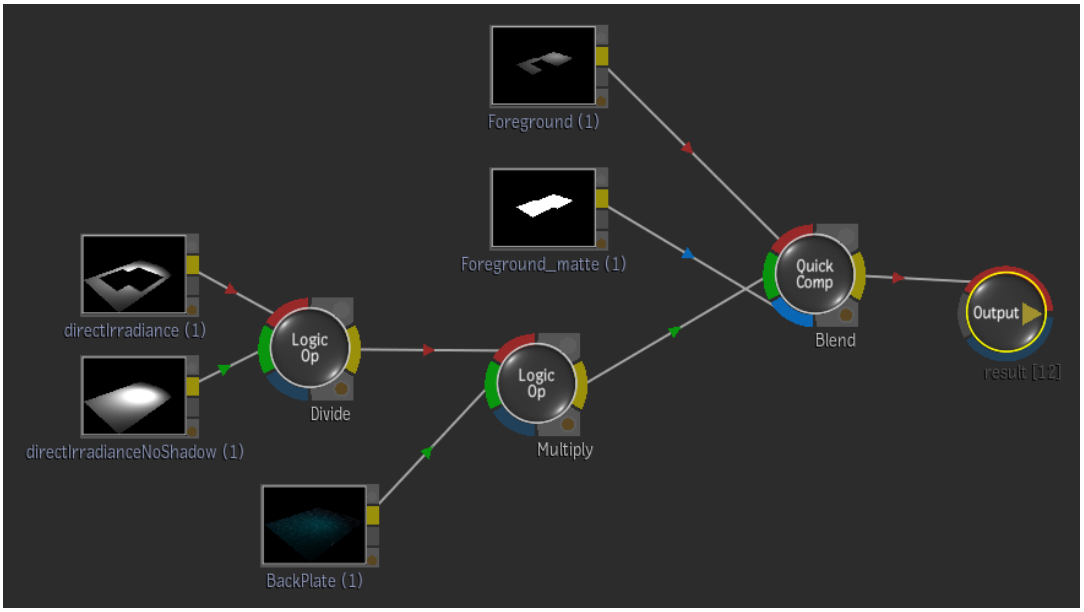


Figure 70 Shadow Re-Projection

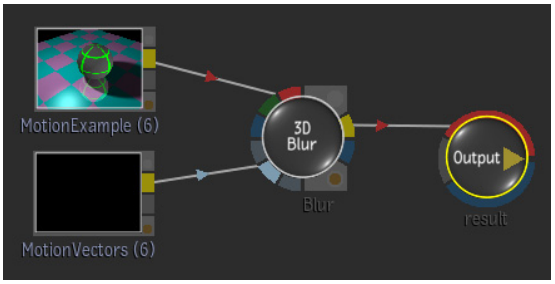


Figure 71 Simple Vector-Based 2D Montion-Blur

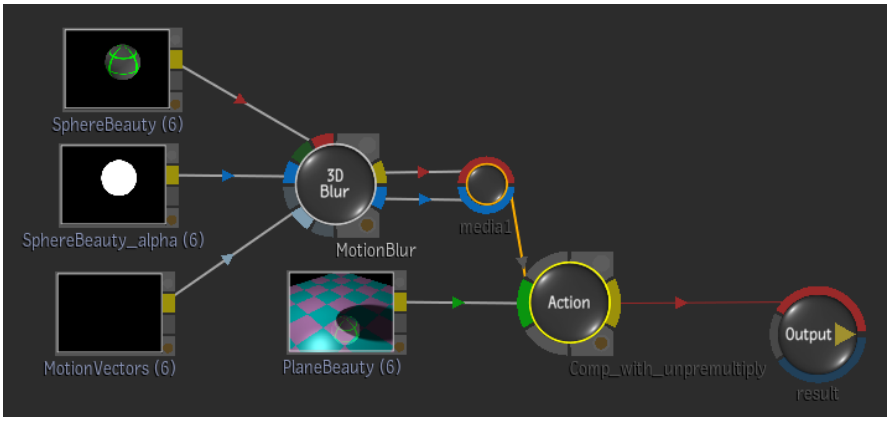


Figure 72 Vector-Based 2D Motion-Blur with Clean Edges

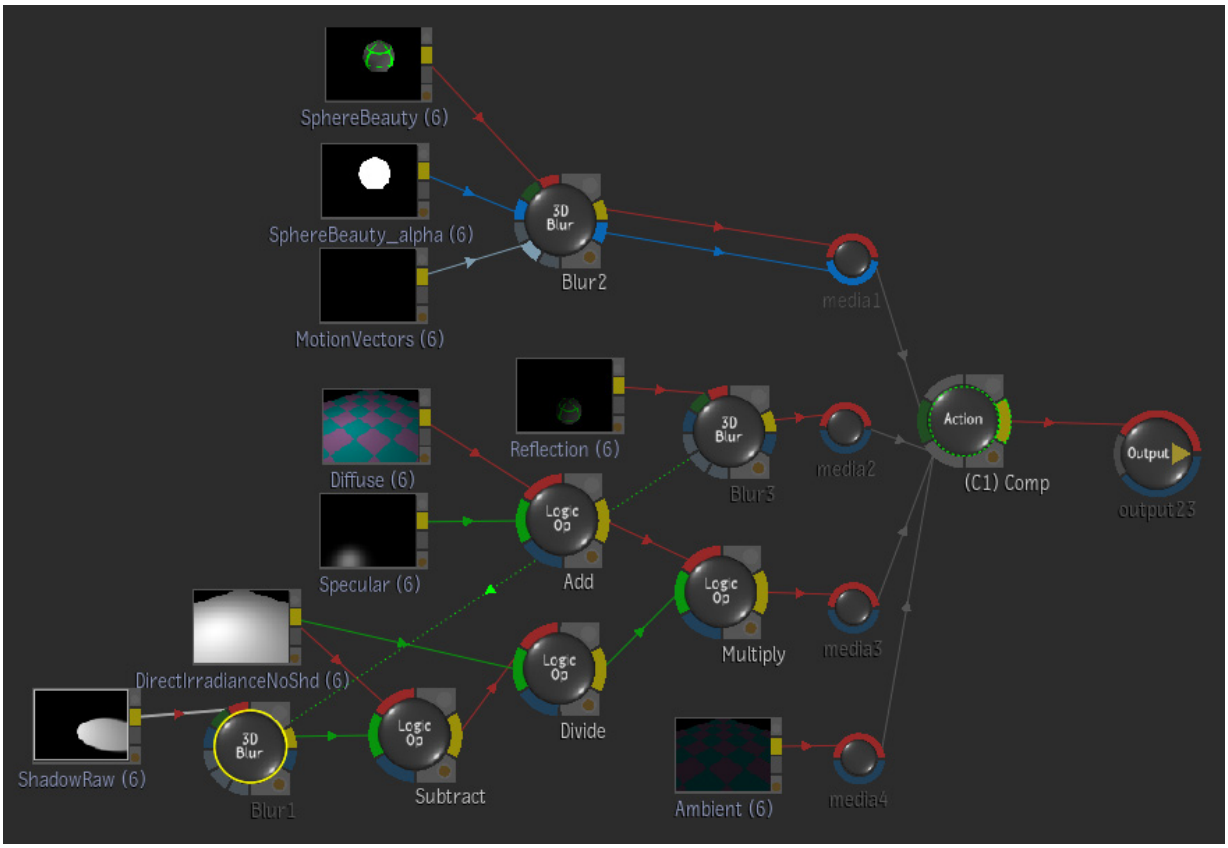


Figure 73 Faking Motion Blur by Component

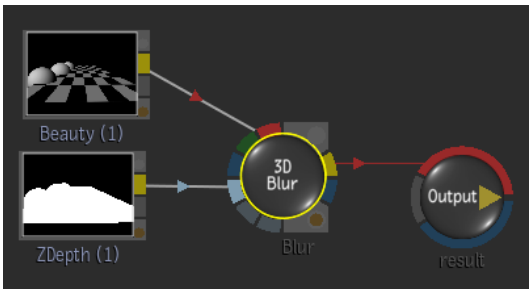


Figure 74 Depth-Based Blur Modulation