mental ray - architectural shaders

Document version 0.95.1
Sept 11, 2006
Copyright Information

Copyright © 1986-2006 mental images GmbH, Berlin, Germany.

All rights reserved.

This document is protected under copyright law. The contents of this document may not be translated, copied or duplicated in any form, in whole or in part, without the express written permission of mental images GmbH.

The information contained in this document is subject to change without notice. mental images GmbH and its employees shall not be responsible for incidental or consequential damages resulting from the use of this material or liable for technical or editorial omissions made herein.

mental images®, incremental images™, mental ray®, mental matter®, mental ray Phenomenon®, mental ray Phenomena™, Phenomenon™, Phenomenon Creator™, Phenomenon Editor™, Photon Map™, mental ray Relay™ Library, Relay™ Library, SPM®, Shape-by-Shading™, Internet Rendering Platform™, iRP™, Reality®, Reality Server®, Reality Player™, Reality Designer™, iray®, imatter®, and neuray™ are trademarks or, in some countries, registered trademarks of mental images GmbH, Berlin, Germany.

All other product names mentioned in this document may be trademarks or registered trademarks of their respective companies and are hereby acknowledged.
# Table of Contents

1 Architectural shader library - introduction ................................................. 1
   1.1 About the library ............................................................................. 1

2 mia_material - material for architectural and design visualisation .......... 3
   2.1 Introduction .................................................................................... 3
      2.1.1 What is the mia_material? ...................................................... 3
      2.1.2 Structure of this Document .................................................... 4
   2.2 Fundamentals .................................................................................. 4
      2.2.1 Physics and the Display ......................................................... 4
         2.2.1.1 A Note on Gamma ........................................................... 4
         2.2.1.2 Tone Mapping ................................................................. 5
      2.2.2 Use Final Gathering and Global Illumination ......................... 5
      2.2.3 Use Physically Correct Lights ................................................ 6
   2.3 Features ......................................................................................... 6
      2.3.1 The Shading Model ................................................................. 6
      2.3.2 Conservation of Energy ......................................................... 7
      2.3.3 BRDF - how Reflectivity Depends on Angle ......................... 8
      2.3.4 Reflectivity Features ............................................................. 9
      2.3.5 Transparency Features .......................................................... 10
         2.3.5.1 Solid vs. Thin-Walled ....................................................... 10
         2.3.5.2 Cutout Opacity ............................................................... 11
      2.3.6 Special Effects ........................................................................ 11
         2.3.6.1 Built in Ambient Occlusion ............................................. 11
      2.3.7 Performance Features ............................................................ 14
   2.4 Material Parameters ....................................................................... 15
      2.4.1 Diffuse ..................................................................................... 15
      2.4.2 Reflections ............................................................................... 15
         2.4.2.1 Basic Features ................................................................ 15
         2.4.2.2 Performance Features .................................................... 18
      2.4.3 Refractions .............................................................................. 19
      2.4.4 Translucency ......................................................................... 21
      2.4.5 Anisotropy ............................................................................. 23
      2.4.6 BRDF .................................................................................... 24
      2.4.7 Special Effects ........................................................................ 25
         2.4.7.1 Built in Ambient Occlusion ............................................. 25
      2.4.8 Advanced Rendering Options ................................................ 27
         2.4.8.1 Reflection Optimization Settings ..................................... 27
2.4.8.2 Refraction Optimization Settings .................................. 28
2.4.8.3 Options ........................................................................ 29
2.4.9 Interpolation ................................................................. 31
2.4.10 Special Maps ............................................................. 34
2.5 Tips and Tricks ............................................................... 36
    2.5.1 Final Gathering Performance ........................................ 36
    2.5.2 Quick Guide to some Common Materials ......................... 36
        2.5.2.1 General Rules of Thumb for Glossy Wood, Flooring, etc. .... 36
        2.5.2.2 Ceramics .......................................................... 37
        2.5.2.3 Stone Materials ............................................... 37
        2.5.2.4 Glass ............................................................ 37
        2.5.2.5 Colored Glass ................................................. 38
        2.5.2.6 Water and Liquids ............................................. 41
        2.5.2.7 The Ocean and Water Surfaces .............................. 43
        2.5.2.8 Metals .......................................................... 45
        2.5.2.9 Brushed Metals .............................................. 46
3 Sun and Sky ................................................................. 49
    3.1 Introduction .............................................................. 49
    3.2 Units ........................................................................... 49
    3.3 Important note on fast SSS and Sun&Sky ........................... 50
    3.4 Common parameters ................................................... 50
    3.5 Sun parameters .......................................................... 52
    3.6 Sky parameters .......................................................... 53
4 Utility shaders .............................................................. 59
    4.1 Round corners ........................................................... 59
    4.2 Tone mapping / Exposure ............................................. 60
5 Advanced topics ......................................................... 63
    5.1 mia_material API ....................................................... 63
        5.1.1 Obtaining sub-components of the rendering ................. 63
        5.1.2 Defining characteristics of light sources ..................... 64
    5.2 mia_material_api.h ..................................................... 64
        5.2.1 Sample shader source ........................................... 66
Chapter 1

Architectural shader library - introduction

1.1 About the library

The *mental ray architectural* library contains a set of shaders designed for architectural and
design visualization.

The most important are the *mia_material*, an easy to use all-around material, and the
*Physical Sun and Sky* shaders, but the library also contains minor tools like shaders to
create render-time rounded corners, and more.

In standalone *mental ray* the shaders are added by including the “mi” declaration file and
linking to the library;

```
link "architectural.dll"
include "architectural.mi"
```

The library strictly requires *mental ray* version 3.5 or newer and will not function on earlier
releases of *mental ray.*
Chapter 2

mia_material - material for architectural and design visualisation

2.1 Introduction

2.1.1 What is the mia_material?

The mental ray mia_material is a monolithic material shader that is designed to support most materials used by architectural and product design renderings. It supports most hard-surface materials such as metal, wood and glass. It is especially tuned for fast glossy reflections and refractions (replacing the DGS material) and high-quality glass (replacing the dielectric material).

The major features are:

- **Easy to use** - yet flexible. Controls arranged logically in a “most used first” fashion.
- **Templates** - for getting faster to reality.
- **Physically accurate** - the material is energy conserving, making shaders that breaks the laws of physics impossible.
- **Glossy performance** - advanced performance boosts including interpolation, emulated glossiness, and importance sampling.
- **Tweakable BRDF\(^1\)** - user can define how reflectivity depends on angle.
- **Transparency** - “Solid” or “thin” materials - transparent objects such as glass can be treated as either “solid” (refracting, built out of multiple faces) or “thin” (non-refracting, can use single faces).

\(^1\)Bidirectional Reflectance Distribution Function
• **Round corners** - shader can simulate “fillets” to allow sharp edges to still catch the light in a realistic fashion.

• **Indirect Illumination control** - set the final gather accuracy or indirect illumination level on a per-material basis.

• **Oren-Nayar diffuse** - allows “powdery” surfaces such as clay.

• **Built in Ambient Occlusion** - for contact shadows and enhancing small details.

• **All-in-one shader** - photon and shadow shader built in.

• **Waxed floors, frosted glass and brushed metals...** - ...all fast and easy to set up.

### 2.1.2 Structure of this Document

This document is divided into sections of *Fundamentals* (beginning on page 4) which explain the main features of the material, the *Parameters* section (page 15) that goes through all the parameters one by one, and a *Tips & Tricks* (page 36) with some advice for users.

### 2.2 Fundamentals

#### 2.2.1 Physics and the Display

The mia_material primarily attempts to be physically accurate hence it has an output with a high dynamic range. How visually pleasing the material looks depends on how the mapping of colors inside the renderer to colors displayed on the screen is done.

When working with the mia_material it is highly encouraged to make sure one is operating through a *tone mapper/exposure control* or at the very least are using gamma correction.

#### 2.2.1.1 A Note on Gamma

Describing all the details about gamma correction is beyond the scope of this document and this is just a brief overview.

The color space of a normal off-the-shelf computer screen is not linear. The color with RGB value 200 200 200 is *not* twice as bright as a color with RGB value 100 100 100 as one would expect.

This is not a “bug” because due to the fact that our eyes see light in a non linear way, the former color is actually perceived to be about twice as bright as the latter. This makes the color space of a normal computer screen roughly perceptually uniform. This is a good thing,
and is actually the main reason 24 bit color (with only 8 bits - 256 discrete levels - for each of
the red, green and blue components) looks as good as it does to our eyes.

The problem is that physically correct computer graphics operate in a true linear color space
where a value represents actual light energy. If one simply maps the range of colors output to
the renderer naively to the 0-255 range of each RGB color component it is incorrect.

The solution is to introduce a mapping of some sort. One of these methods is called gamma
correction.

Most computer screens have a gamma of about 2.2, but most software default to a gamma
of 1.0, which makes everything (especially midtones) look too dark, and light will not “add
up” correctly.

Using gamma of 2.2 is the theoretically “correct” value, making the physically linear light
inside the renderer appear in a correct linear manner on screen.

However, since the response of photographic film isn’t linear either, users have found this
“theoretically correct” value looks too “bright” and “washed out”, and a very common
compromise is to render to a gamma of 1.8, making things look more “photographic”, i.e.
as if the image had been shot on photographic film and then developed.

2.2.1.2 Tone Mapping

Another method to map the physical energies inside the renderer to visually pleasing pixel
values is known as tone mapping. This can be done either by rendering to a floating point file
format and using external software, or use some plugin to the renderer to do it on-the-fly.

A very simple tone mapping shader is included in the library named mia_exposure_simple
and is documented on page 60

2.2.2 Use Final Gathering and Global Illumination

The material is designed to be used in a realistic lighting environment, i.e. using full direct
and indirect illumination.

In mental ray there are two basic methods to generate indirect light: Final Gathering and
Global Illumination. For best results at least one of these methods should be used.

At the very least one should enable Final Gathering, or use Final Gathering combined with
Global Illumination (photons) for quality results. Performance tips for using Final Gather
and Global Illumination can be found on page 36 of this document.

If you are using an environment for your reflections, make sure the same environment (or a
blurred copy of it) is used to light the scene through Final Gathering.

\footnote{This is also known as the “sRGB” color space}
2.2.3 Use Physically Correct Lights

Traditional computer graphics light sources live in a cartoon universe where the intensity of the light doesn’t change with the distance. The real world doesn’t agree with that simplification. Light decays when leaving a light source due to the fact that light rays diverge from their source and the “density” of the light rays change over distance. This decay of a point light source is $1/d^2$, i.e. light intensity is proportional to the inverse of the square of the distance to the source.

One of the reasons for this traditional oversimplification is actually the fact that in the early days of computer graphics tone mapping was not used and problems of colors “blowing out” to white in the most undesirable ways\(^3\) was rampant.

However, as long as only Final Gathering (FG) is used as indirect illumination method, such traditional simplifications still work. Even light sources with no decay still create reasonable renderings! This is because FG is only concerned with the transport of light from one surface to the next, not with the transport of light from the light source to the surface.

It’s when working with Global Illumination (GI) (i.e. with photons) the troubles arise.

When GI is enabled, light sources shoot photons. It is imperative for the mia\textunderscore material (or any other mental ray material) to work properly for the energy of these photons to match the direct light cast by that same light! And since photons model light in a physical manner, decay is “built in”.

Hence, when using GI:

- Light sources must be emitting photons at the correct energy
- The direct light must decay in a physically correct way to match the decay of the photons.

Therefore it is important to make sure the light shader and the photon emission shader of the lights work well together.

2.3 Features

2.3.1 The Shading Model

From a usage perspective, the shading model consists of three components:

- **Diffuse** - diffuse channel /including Oren Nayar “roughness”)/

\(^3\)Raw clipping in sRGB color space is very displeasing to the eye, especially if one color channel clips earlier than the others. Tone mapping generally solves this by “soft clipping” in a more suitable color space than sRGB.
• **Reflections** - glossy anisotropic reflections (and highlights).

• **Refractions** - glossy anisotropic transparency (and translucency).

Direct and indirect light from the scene both cause diffuse reflections as well as translucency effects. Direct light sources also cause traditional “highlights” (specular highlights).

Raytracing is used to create reflective and refractive effects, and advanced importance-driven multi-sampling is used to create glossy reflections and refractions.

The rendering speed of the glossy reflections/refractions can further be enhanced by interpolation as well as “emulated” reflections with the help of Final Gathering.

### 2.3.2 Conservation of Energy

One of the most important features of the material is that it is *automatically energy conserving*. This means that it makes sure that \( \text{diffuse} + \text{reflection} + \text{refraction} \leq 1 \), i.e. that no energy is magically created and the incoming light energy is properly distributed to the diffuse, reflection and refraction components in a way that maintains the first law of thermodynamics\(^4\).

In practice, this means for example that when adding more reflectivity, the energy must be taken from somewhere, and hence the diffuse level and the transparency will be automatically reduced accordingly. Similarly, when adding transparency, this will happen at the cost of the diffuse level.

\(^4\)The first law of thermodynamics is that no one talks about thermodynamics ;)

The *mia_material shading model*
The rules are as follows:

- Transparency takes energy from Diffuse, i.e. at 100% transparency, there will be no diffuse at all.
- Reflectivity takes energy from both Diffuse and Transparency, i.e. a 100% reflectivity there will be neither diffuse nor transparency.
- Translucency is a type of transparency, and `refr_trans_w` defines the percentage of transparency vs. translucency.

\[\text{From left to right: Reflectivities 0.0, 0.4, 0.8 and 1.0}\]

\[\text{From left to right: Transparencies 0.0, 0.4, 0.8 and 1.0}\]

It also means that the level of highlights is linked to the glossiness of a surface. A high `refl_gloss` value causes a narrower but very intense highlight, and a lower value causes a wider but less intense highlight. This is because the energy is now spread out and dissipated over a larger solid angle.

### 2.3.3 BRDF - how Reflectivity Depends on Angle

In the real world, the reflectivity of a surface is often view angle dependent. A fancy term for this is BRDF (Bidirectional Reflectance Distribution Function), i.e. a way to define how much a material reflects when seen from various angles.
Many materials exhibit this behaviour. Glass, water and other dielectric materials with fresnel effects (where the angular dependency is guided strictly by the Index of Refraction) are the most obvious examples, but other layered materials such as lacquered wood, plastic, etc. display similar characteristics.

The mia_material allows this effect both to be defined by the Index of Refraction, and also allows an explicit setting for the two reflectivity values for:

- 0 degree faces (surfaces directly facing the camera)
- 90 degree faces (surfaces 90 degrees to the camera)

See the BRFD section on page 24 for more details.

2.3.4 Reflectivity Features

The final surface reflectivity is in reality caused by the sum of three components:

- The Diffuse effect
- The actual reflections
- Specular highlights that simulate the reflection of light sources
In the real world “highlights” are just (glossy) reflections of the light sources. In computer graphics it’s more efficient to treat these separately. However, to maintain physical accuracy the material automatically keeps “highlight” intensity, glossiness, anisotropy etc. in sync with the intensity, glossiness and anisotropy of reflections, hence there are no separate controls for these as both are driven by the reflectivity settings.

2.3.5 Transparency Features

The material supports full glossy anisotropic transparency, as well as includes a translucent component, described more in detail on page 21.

Translucency

2.3.5.1 Solid vs. Thin-Walled

The transparency/translucency can treat objects either as solid or thin walled.

If all objects were treated as solids at all times, every single window pane in an architectural model would have to be modelled as two faces; an entry surface (that refracts the light slightly in one direction), and immediately following it an exit surface (where the light would be refracted back into the original direction).

Not only is this additional modelling work, it is a waste of rendering power to model a refraction that has very little net effect on the image. Hence the material allows modelling the entire window pane as one single flat plane, foregoing any actual “refraction” of light.
In the above image the helicopter canopy, the window pane, the translucent curtain and the right sphere all use “thin walled” transparency or translucency, whereas the glass goblet, the plastic horse and the left sphere all use “solid” transparency or translucency.

2.3.5.2 Cutout Opacity

Beyond the “physical” transparency (which models an actual property of the material) there is a completely separate non-physical “cutout opacity” channel to allow “billboard” objects such as trees, or to cut out things like a chainlink fence with an opacity mask.

2.3.6 Special Effects

2.3.6.1 Built in Ambient Occlusion

*Ambient Occlusion* (henceforth referred to as “AO”) is a method spearheaded by the film industry to emulate the “look” of true global illumination by using shaders that calculate how occluded (i.e. blocked) an area is from receiving incoming light.
Used alone, an AO shader\textsuperscript{5} creates a grayscale output that is “dark” in areas to which light cannot reach and “bright” in areas where it can:

As seen in the above image, one of the main results of AO is dark in crevices and areas where light is blocked by other surfaces and it is bright in areas that are exposed to the environment.

One important aspect of AO is that one can tune the “distance” within which it looks for occluding geometry.

\textsuperscript{5}Like the separate mental ray \texttt{mib\_amb\_occlusion} shader
2.3 Features

Using a radius creates only a “localized” AO effect; only surfaces that are within the given radius are actually considered occluders (which is also massively faster to render). The practical result is that the AO gives us nice “contact shadow” effects and makes small crevices visible.

There are two ways to utilize the built in AO in the mia.material:

- “Traditional” AO for adding an omnipresent ambient light that is then attenuated by the AO to create details.
- Use AO for detail enhancement together with existing indirect lighting methods (such as Final Gathering or photons).

The latter method is especially interesting when using a highly “smoothed” indirect illumination solution (i.e. a very high photon radius, or an extremely low final gather density) which could otherwise lose small details. By applying the AO with short rays these details can be brought back.
2.3.7 Performance Features

Finally the mia_material contains a large set of built in functions for top performance, including but not limited to:

- Advanced importance sampling with ray rejection thresholds
- Adaptive glossy sample count
- Interpolated glossy reflection/refraction with detail enhancements
- Ultra fast emulated glossy reflections (refl_hl_only mode)
- Possibility to ignore internal reflections for glass objects
- Allowing a choice between traditional transparent shadows (suitable for e.g. a window pane) and refractive caustics (suitable for solid glass objects) on a per material basis.
2.4 Material Parameters

2.4.1 Diffuse

diffuse_weight sets the desired level (and diffuse the color) of the diffuse reflectivity. Since the material is energy conserving, the actual diffuse level used depends on the reflectivity and transparency as discussed above.

The diffuse component uses the Oren-Nayar shading model. When diffuse_roughness is 0.0 this is identical to classical Lambertian shading, but with higher values the surface gets a a more “powdery” look:

![Roughness 0.0 (left), 0.5 (middle) and 1.0 (right)](image)

2.4.2 Reflections

2.4.2.1 Basic Features

The reflectivity and refl_color together define level of reflections as well as the intensity of the traditional “highlight” (also known as “specular highlight”).

This value is the maximum value - the actual value also depends on the angle of the surface and come from the BRDF curve. This curve (described in more detail on page 24) allows one to define a brdf_0_degree_refl (for surfaces facing the view) and brdf_90_degree_refl (for surfaces perpendicular to the view).
• The left cup shows no reflectivity at all and a purely diffuse material.

• The center cup shows a \texttt{brdf\_0\_degree\_refl} of 0.1 and a \texttt{brdf\_90\_degree\_refl} of 1.0.

• The right cup has a both a \texttt{brdf\_0\_degree\_refl} and \texttt{brdf\_90\_degree\_refl} of 0.9, i.e. constant reflectivity across the surface.

Note how the high reflectivity automatically “subtracts” from the white diffuse color. If this didn’t happen, the material would become unrealistically over-bright, and would break the laws of physics.\footnote{See page 7}

The \texttt{refl\_gloss} parameter defines the surface “glossiness”, ranging from 1.0 (a perfect mirror) to 0.0 (a diffusely reflective surface):
The `refl_samples` parameters defines the maximum\(^7\) number of samples (rays) are shot to create the glossy reflections. Higher values renders slower but create a smoother result. Lower values render faster but create a grainier result. Generally 32 is enough for most cases.

There are two special cases:

- Since a `refl_gloss` value of 1.0 equals a “perfect mirror” it is meaningless to shoot multiple rays for this case, hence only one reflection ray is shot.

- If the `refl_samples` value is set to 0, the reflections will be “perfect mirror” (and only one ray shot) regardless of the actual value of `refl_gloss`. This can be used to boost performance for surfaces with very weak reflections. The highlight still obeys the glossiness value.

Metallic objects actually influence the color of their reflection whereas other materials do not. For example, a gold bar will have gold colored reflections, whereas a red glass orb does not have red reflections. This is supported through the `refl_is_metal` option.

- When off, the `refl_color` parameter defines the color and `reflectivity` parameter (together with the BRDF settings) the intensity and colors of reflections.

- When on, the `diffuse` parameter defines the color of reflections, and `reflectivity` parameter sets the “weight” between diffuse reflections and glossy (metallic) reflections.

\(^7\)The actual number is adaptive and depends on reflectivity, ray importance, and many other factors.
The left image shows non-metallic reflections (refl_is_metal is off). One can see reflections clearly contain the color of the objects they reflect and are not influenced by the color of the materials.

The center image uses metallic reflections (refl_is_metal is on). Now the color of reflections are influenced by the color of the object. The right image shows a variant of this with the reflectivity at 0.5, creating a 50:50 mix between colored reflections and diffuse reflections.

2.4.2.2 Performance Features

Glossy reflections need to trace multiple rays to yield a smooth result, which can become a performance issue. For this reason there are a couple of special features designed to enhance their performance.

The first of those features is the interpolation. By turning refl_interpolate) on, a smoothing algorithm allows rays to be re-used and smoothed. The result is faster and smoother glossy reflections at the expense of accuracy. Interpolation is explained in more detail on page 31.

For highly reflective surfaces it is clear that true reflection rays are needed. However, for less reflective surfaces (where it is less “obvious” that the surface is really reflecting anything) there exists a performance-enhancing shortcut, the refl_hi_only switch.

When refl_hi_only is on, no actual reflection rays are traced. Instead only the “highlights” are shown, as well as soft reflections emulated with the help of using Final Gathering.  

8The technique works best on flat surfaces  
9If Final Gathering is not enabled, this mode simply shows the highlights and attempts no emulation of reflections.
The `refl_hl_only` mode takes no additional render time compared to a non-glossy (diffuse) surface, yet can yield surprisingly convincing results. While it may not be completely convincing for “hero” objects in a scene it can work very well for less essential scene elements. It tends to work best on materials with weak reflections or extremely glossy (blurred) reflections:

![The left two cups use real reflections, those on the right use refl_hl_only](image)

While the two cups on the left are undoubtedly more convincing than those on the right, the fact that the right hand cups have no additional render time compared to a completely non-reflective surface makes this mode very interesting. The emulated reflections still pull in a directional color bleed such that the bottom side of the cup is influenced by the color of the wooden floor just as if it was truly reflective.

### 2.4.3 Refractions

The `transparency` parameter defines the level of refractions and `refr_color` defines the color. While this color can be used to create “colored glass”, there is a slightly more accurate method to do this described on page 37.

Due to the materials energy conserving nature (see page 7) the value set in the `transparency` parameter is the maximum value - the actual value depends on the reflectivity as well as the BRDF curve.

The `refr_ior` defines the Index of Refraction, which is a measurement of how much a ray of light “bends” when entering a material. Which direction light bends depends on if it is entering or exiting the object. The `mia_material` use the direction of the surface normal as the primary cue for figuring out whether it is entering or exiting. It is therefore important to model transparent refractive objects with the surface normal pointing in the proper direction.

The IOR can also be used to define the BRDF curve, which is what happens in the class of transparent materials known as “dielectric” materials, and is illustrated here:
Note how the leftmost cup looks completely unrealistic and is almost invisible. Because an IOR of 1.0 (which equals that of air) is impossible, we get no change in reflectivity across the material and hence perceive no “edges” or change of any kind. Whereas the middle and rightmost cups have a realistic change in reflectivity guided by the IOR.

One is however not forced to base the reflectivity on the IOR but can instead use the BRDF mode to set it manually:

The left cup again acquires it’s curve from the index of refraction. The center cup has a manually defined curve, which has been set to a \texttt{brdf\_90\_degree\_refl} of 1.0 and a \texttt{brdf\_0\_degree\_refl} of 0.2, which looks a bit more like metallized glass. The rightmost cup uses the same BRDF curve, but instead is set to “thin walled” transparency (see page 10).
2.4 Material Parameters

Clearly, this method is the better way to make “non-refractive” objects compared to simply setting `refr_iор` to 1.0 as we tried above.

As with reflections, the `refr_gloss` parameter defines how sharp or blurry the refractions/transparency are, ranging from a 1.0 (a completely clear transparency) to 0.0 (an extremely diffuse transparency):

![A refr_gloss of 1.0 (left), 0.5 (center) and 0.25 (right)](image)

Just as with the glossy reflections, the glossy transparency has a `refr_interpolate` switch, allowing faster, smoother, but less accurate glossy transparency. Interpolation is described on page 31.

2.4.4 Translucency

Translucency is handled as a special case of transparency, i.e. to use translucency there must first exist some level of transparency, and the `refr_trans_w` parameter decides how much of this is used as transparency and how much is translucency:
A *transparency* of 0.75 and a *refr_trans_w* of 0.0 (left), 0.5 (center) and 1.0 (right).

- If *refr_trans_w* is 0.0, all of the *transparency* is used for transparency.
- If *refr_trans_w* is 0.5, half of the *transparency* is used for transparency and half is used for translucency.
- If *refr_trans_w* is 1.0, all of the *transparency* is used for translucency and there is *no* actual transparency.

The translucency is primarily intended to be used in “thin walled” mode (as in the example above) to model things like curtains, rice paper, or such effects. In “thin walled” mode it simply allows the shading of the reverse side of the object to “bleed through”.

The shader also operates in “Solid” mode, but the implementation of translucency in the *mia_material* is a simplification concerned solely with the transport of light from the back of an object to it’s front faces and is not “true” SSS (sub surface scattering). An “SSS-like” effect can be generated by using glossy transparency coupled with translucency but it is neither as fast nor as powerful as the dedicated SSS shaders.
2.4.5 Anisotropy

Anisotropic reflections and refractions can be created using the **anisotropy** parameter. The parameter sets the ratio between the “width” and the “height” of the highlights, hence when **anisotropy** is 1.0 there is no anisotropy, i.e. the effect is disabled.

For other values of **anisotropy** (above and below 1.0 are both valid) the “shape” of the highlight (as well as the appearance of reflections) change.

The anisotropy can be rotated by using the **anisotropy_rotation** parameter. The value 0.0 is unrotated, and the value 1.0 is one full revolution (i.e. 360 degrees). This is to aid using a
texture map to steer the angle:

![Image](image-url)

*anisotropy_rotation* values of 0.0 (left), 0.25 (center) and textured (right)

**Note:** When using a textured *anisotropy_rotation* it is important that this texture is *not* anti-aliased (filtered). Otherwise the anti-aliased pixels will cause local vortices in the anisotropy that appear as seam artifacts.

If the *anisotropy_channel* parameter is -1 the base rotation follows the local object coordinate system. Otherwise, the space which defines the “stretch directions” of the highlights are derived from the texture space set by *anisotropy_channel*\(^{10}\).

See also “brushed metal” on page 46 in the tips section.

### 2.4.6 BRDF

As explained in the introduction on page 8 the materials reflectivity is ultimately guided by the *incident angle* from which it is viewed.

\(^{10}\)Note that deriving the anisotropy from texture space only creates one space per triangle and may cause visible seams between triangles.
There are two modes to define this BRDF curve:

The first mode is “by IOR”, i.e. when \texttt{brdf\_fresnel} is on. How the reflectivity depends on the angle is then solely guided by the materials IOR. This is known as \textit{fresnel reflections} and is the behaviour of most dielectric materials such as water, glass, etc.

The second mode is the manual mode, when \texttt{brdf\_fresnel} is off. In this mode the \texttt{brdf\_0\_degree\_refl} parameter defines the reflectivity for surfaces directly facing the viewer (or incident ray), and \texttt{brdf\_90\_degree\_refl} defines the reflectivity of surfaces perpendicular to the viewer. The \texttt{brdf\_curve} parameter defines the falloff of this curve.

This mode is used for most hybrid materials or for metals. Most material exhibit strong reflections at grazing angles and hence the \texttt{brdf\_90\_degree\_refl} parameter can generally be kept at 1.0 (and using the \texttt{reflectivity} parameter to guide the overall reflectivity instead). Metals tend to be fairly uniformly reflective and the \texttt{brdf\_0\_degree\_refl} value is high (0.8 to 1.0) but many other layered materials, such as linoleum, lacquered wood, etc. has lower \texttt{brdf\_0\_degree\_refl} values (0.1 - 0.3).

See the tips on page 36 for some guidelines.

2.4.7 Special Effects

2.4.7.1 Built in Ambient Occlusion

The built in Ambient Occlusion (henceforth shortened to “AO”) can be used in two ways. Either it is used to enhance details and “contact shadows” in indirect illumination (in which case there must first \textit{exist} some form of indirect illumination in the first place), or it is used together with a specified “ambient light” in a more traditional manner. Hence, if neither
indirect light exists, nor any “ambient light” is specified, the AO will have no effect 11.

The **ao_samples** sets the number of samples (rays) shot for creating the AO. Higher value is smoother but slower, lower values faster but grainier. 16 is the default and 64 covers most situations.

The **ao_distance** parameter defines the radius within which occluding objects are found. Smaller values restrict the AO effect only to small crevices but are much faster to render. Larger values cover larger areas but render slower. The following images illustrate the raw AO contribution with two different distances:

The **ao** do **details** switch.

This mode is used to apply short distance AO multiplying it with the existing *indirect illumination* (Final Gathering or GI/photons), bringing out small details.

Study this helicopter almost exclusively lit by indirect light:

---

11 Sometimes people use AO as a general multiplier to all diffuse light. This has the distinct drawback of affecting even brightly *directly* lit areas with “AO shadows”, which can look wrong. This use is not covered by the built in AO shader because it is trivially acheived by simply applying the mib_amb_occlusion shader to the diffuse color of the material and putting the materials original color into it’s Bright parameter.
Note how the helicopter does not feel “grounded” in the left image and the shadows under the landing skids are far too vague. The right image uses AO to “punch out” the details and the contact shadows.

The **ao.dark** parameter sets the “darkness” of the AO shadows. It is used as the multiplier value for completely occluded surfaces. In practice this means: A black color will make the AO effect very dark, a middle gray color will make the effect less noticeable (brighter) etc.

The **ao.ambient** parameter is used for doing more “traditional” AO, i.e. supplying the imagined “ever present ambient light” that is then attenuated by the AO effect to create shadows.

While “traditional AO” is generally used when rendering *without* other indirect light, it can also be combined with existing indirect light. One needs to keep in mind that this magical “ever present ambient light” is inherently non-physical, but may perhaps help lighten some troublesome dark corners.

### 2.4.8 Advanced Rendering Options

#### 2.4.8.1 Reflection Optimization Settings

These parameters define some performance boosting options for reflections.

**refl_falloff_dist** allows limiting reflections to a certain distance, which both speeds up rendering as well as avoiding pulling in distant objects into extremely glossy reflections.

If **refl_falloff_color** is enabled and used, reflections will fade to this color. If it is not enabled, reflections will fade to the environment color. The former tends to be more useful for indoor scenes, the latter for outdoor scenes.
Each material can locally set a maximum trace depth using the `refl_depth` parameter. When this trace depth is reached the material will behave as if the `refl_hl_only` switch was enabled, i.e. only show highlights and “emulated” reflections. If `refl_depth` is zero, the global trace depth is used.

`refl_cutoff` is a threshold at which reflections are rejected (not traced). It’s a relative value, i.e. the default of 0.01 means that rays that contribute less than 1% to the final pixel are ignored.

### 2.4.8.2 Refraction Optimization Settings

The optimization settings for refractions (transparency) are nearly identical to those for reflections. The exception is that of `refr_falloff_color` which behaves differently.

- When `refr_falloff_dist` is used, and `refr_falloff_color` is not used, transparency rays will fade to black. This is like smoked glass or highly absorbent materials. Transparency will just completely stop at a certain distance. This has the same performance advantage as using the `refl_falloff_dist` for reflections, i.e. tracing shorter rays are much faster.

- However, when `refr_falloff_color` is used, it works differently. The material will then make physically correct absorption. Exactly at the distance given by `refr_falloff_dist` will the refractions have the color given by `refr_falloff_color` - but the rays are not limited in reach. At twice the distance, the influence of `refr_falloff_color` is double, at half the distance half, etc.

The leftmost cup has no fading. The center cup has `refr_falloff_color` off, and hence fades to black, which also includes the same performance benefits of limiting the trace distance as when used for reflections.
The rightmost cup, however, fades to a blue color. This causes proper exponential attenuation in the material, such that the thicker the material, the deeper the color. See page 37 for a discussion about realistic colored glass.

**Note:** To render proper shadows when using `refr_falloff_dist` one must use ray traced shadows, and the shadow mode must be set to `segment`. See the mental ray manual on shadow modes.

`refl_depth` and `refl_cutoff` works identical to the reflection case described above.

### 2.4.8.3 Options

The options contain several on/off switches that control some of the deepest details of the material:

The `thin_walled` decides if a material causes refractions (i.e. behaves as if it is made of a solid transparent substance) or not (i.e. behaves as if made of wafer-thin sheets of a transparent material). This topic is discussed in more detail on page 10.

![Solid (left) and Thin-walled (right)](image)

The `do_refractive_caustics` parameter defines how glass behaves when `caustics` are enabled.

When not rendering caustics, the `mia_material` uses a shadow shader to create transparent shadows. For objects such as window panes this is perfectly adequate, and actually creates a better result than using caustics since the direct light is allowed to pass (more or less) undisturbed through the glass into e.g. a room.

Traditionally, enabling caustics in `mental ray` cause all materials to stop casting transparent shadows and instead start to generate refractive caustics. In most architectural scenes this is undesirable; one may very well want a glass decoration on a table to generate caustic effect,
but still want the windows of the room to let in quite normal direct light. This switch makes this possible on the material level.

Using transparent shadows

Using refractive caustics

The left image shows the result that happen when \texttt{do_refractive_caustics} is off, the right the result when it is on. Both modes can be freely mixed within the same rendering. Photons are automatically treated accordingly by the built in photon shader, shooting straight through as direct light in the former case, and being refracted as caustics in the latter.

The \texttt{backface_cull} switch enables a special mode which makes surfaces completely invisible to the camera when seen from the reverse side. This is useful to create “magic walls” in a room. If all walls are created as planes with the normal facing inwards, the \texttt{backface_cull} switch allows the room to be rendered from “outside”. The camera will see into the room, but the walls will still “exist” and cast shadows, bounce photons, etc. while being magically “see through” when the camera steps outside.

No Backface Culling

Backface Culling on the walls

The \texttt{propagate_alpha} switch defines how transparent objects treats any alpha channel information in the background. When on, refractions and other transparency effects will propagate the alpha of the background “through” the transparent object. When off, transparent objects will have an opaque alpha.

The \texttt{no_visible_area_hl} parameter concerns the behaviour of visible area lights.
Keep in mind that traditional “highlights” (i.e. specular effects) is a computer graphics “trick” in place of actually creating a glossy reflection of an actual visible light-emitting surface.

However, mental ray area lights can be visible, and when they are visible they will reflect in any (glossy) reflective objects. If both the reflection of the visible area light and the highlight is rendered, the light is added twice, causing an unrealistic brightening effect. This switch (which defaults to on) causes visible area lights to loose their “highlights” and instead only appear as reflections\textsuperscript{12}.

A final optimization switch (also on by default) is the skip_inside_refl checkbox. Most reflections on the insides of transparent objects are very faint, except in the special case that occurs at certain angles known as “Total Internal Reflection” (TIR). This switch saves rendering time by ignoring the weak reflections completely but retaining the TIR’s.

The indirect_multiplier allows tweaking of how strongly the material responds to indirect light, and fg_quality is a local multiplier for the number of final gather rays shot by the material. Both default to 1.0 which uses the global value.

To aid in mapping textures to fg_quality the additional fg_quality_w parameter exists. When zero, fg_quality is the raw quality setting, but for a nonzero fg_quality_w the actual quality used is the product of the two values, with a minimum of 1.0. This means that with a color texture mapped to fg_quality and fg_quality_w set to 5.0, black in the texture results in a quality of 1.0 (i.e. the number of final gather rays shot is the global default), and white in the texture in a quality of 5.0 (five times as many rays are shot).

2.4.9 Interpolation

Glossy reflections and refractions can be interpolated. This means they render faster and become smoother.

Interpolation works by precalculating glossy reflection in a grid across the image. The number of samples (rays) taken at each point is governed by the refl_samples or refr_samples parameters just as in the non-interpolated case. The resolution of this grid is set by the intr_grid_density parameter.

However, interpolation can cause artifacts. Since it is done on a low resolution grid, it can lose details. Since it blends neighbours of this low resolution grid it can cause over smoothing. For this reason it is primarily useful on flat surfaces. Wavy, highly detailed surfaces, or surfaces using bump maps will not work well with interpolation.

Valid values for intr_grid_density parameter is:

- 0 = grid resolution is double that of the rendering
- 1 = grid resolution is same as that of the rendering

\textsuperscript{12}Naturally this does not apply to the refl_hl_only mode, since it doesn’t actually reflect anything
• 2 = grid resolution is half of that of the rendering
• 3 = grid resolution is a third of that of the rendering.
• 4 = grid resolution is a fourth of that of the rendering.
• 5 = grid resolution is a fifth of that of the rendering.

Within the grid data is stored and shared across the points. Lower grid resolutions is faster but lose more detail information. Both reflection and refraction has an intr_refl_samples parameter which defines how many stored grid points (in an N by N group around the currently rendered point) is looked up to smooth out the glossiness. The default is 2, and higher values will “smear” the glossiness more, but are hence prone to more overmoothing artifacts.

The reflection of the left cup in the floor is not using interpolation, and one can perceive some grain (here intentionally exaggerated). The floor tiles under the other two cup uses a half resolution interpolation with 2 (center) and 4 (right) point lookup respectively.

This image also illustrates one of the consequences of using interpolation: The foot of the left cup, which is near the floor, is reflected quite sharply, and only parts of the cup far from the floor are blurry. Whereas the interpolated reflections on the right cups have a certain “base level” of blurriness (due to the smoothing of interpolation) which makes even the closest parts somewhat blurry. In most scenes with weak glossy reflections this discrepancy will never be noticed, but in other cases this can make things like legs of tables and chairs feel “unconnected” with a glossy floor, if the reflectivity is high.

To solve this the intr_refl_ddist parameter exists. It allows a second set of detail rays to be traced to create a “clearer” version of objects within that radius.
All three floor tiles use interpolation but the rightmost two use different distances for the “detail distance”.

This also allows an interesting “trick”: Set the `refl_samples` to 0, which renders reflections as if they were mirror-perfect but use the interpolation to introduce blur into this “perfect” reflection (and perhaps use the `intr_refl_ddist` to make nearby parts less blurry). This is an extremely fast way to obtain a glossy reflection.

The above floor tiles are rendered with mirror reflections, and the “blurriness” comes solely from the interpolation. This renders as fast (or faster!) than pure mirror reflections, yet gives a satisfying illusion of true glossy reflections, especially when utilizing the `intr_refl_ddist` as on the right.
2.4.10 Special Maps

The mia_material also supports the following special inputs:

The **bump** accepts a shader that perturbs the normal for bump mapping.

When **no_diffuse_bump** is *off*, the bumps apply to all shading components (diffuse, highlights, reflections, refractions...). When it is *on*, bumps are applied to all component except the diffuse. This means bumps are seen in reflections, highlights, etc. but the diffuse shading shows no bumps. It is as if the materials diffuse surface is smooth, but covered by a bumpy lacquer coating.

![](image)

**no_diffuse_bump** is off (left) and on (right)

The **cutout_opacity** is used to apply an opacity map to completely remove parts of objects. A classic example is to map an image of a tree to a flat plane and use opacity to cut away the parts of the tree that are not there.
Mapping the transparency (left) vs. cutout opacity (right)

The additional_color is an input to which one can apply any shader. The output of this shader is simply added on top of the shading done by the mia_material and can be used both for “self illumination” type effects as well as adding whatever additional shading one may want.

The material also supports standard displacement and environment shaders. If no environment is supplied, the global camera environment is used.
2.5 Tips and Tricks

2.5.1 Final Gathering Performance

The Final Gathering algorithm in mental ray 3.5 is vastly improved from earlier versions, especially in its adaptivity. This means one can often use much lower ray counts and much lower densities than in previous versions of mental ray.

Many stills can be rendered with such extreme settings as 50 rays and a density of 0.1 - if this causes “over smoothing” artifacts, one can use the built in AO (see page 25) to solve those problems.

When using Final Gathering together with GI (photons), make sure the photon solution is fairly “smooth” by rendering with Final Gathering disabled first. If the photon solution is noisy, increase the photon search radius until it “calms down”, and then re-enable Final Gathering.

2.5.2 Quick Guide to some Common Materials

Here are some quick rules-of-thumb for creating various materials. They each assume basic default settings as a starting point.

2.5.2.1 General Rules of Thumb for Glossy Wood, Flooring, etc.

This is the kind of “hybrid” materials one run into in many architectural renderings; lacquered wood, linoleum, etc.

For these materials brdf, fresnel should be off (i.e. we define a custom BRDF curve). Start out with brdf_0_degree_refl of 0.2 and brdf_90_degree_refl of 1.0 and apply some suitable texture map to the diffuse. Set reflectivity around 0.5 to 0.8.

How glossy is the material? Is reflections very clear or very blurry? Are they Strong or Weak?

- For clear, fairly strong reflections, keep refl_gloss at 1.0
- For slightly blurry but strong reflections, set a lower refl_gloss value. If performance becomes an issue try using refl_interpolate).
- For slightly blurry but also very weak reflections one can often “cheat” by setting a lower refl_gloss value (to get the broader highlights) but set refl_samples value to 0. This shoots only one mirror ray for reflections - but if they are very weak, one can often not really tell.
- For medium blurry surfaces set an even lower refl_gloss and maybe increase the refl_samples. Again, for performance try refl_interpolate).
- For *extremely* blurry surfaces or surfaces with very weak reflections, try using the `refl_hl_only` mode.

A typical wooden floor could use `refl_gloss` of 0.5, `refl_samples` of 16, `reflectivity` of 0.75, a nice wood texture for `diffuse`, perhaps a slight bump map (try the `no_diffuse_bump` checkbox if bumpiness should appear only in the lacquer layer).

A linoleum carpet could use the same but with a different texture and bump map, and probably with a slightly lower `reflectivity` and `refl_gloss`.

### 2.5.2.2 Ceramics

Ceramic materials are *glazed*, i.e. covered in a thin layer of transparent material. They follow similar rules to the general materials mentioned above, but one should have `brdf_fresnel` *on* and the `refr_ior`) set at about 1.4 and `reflectivity` at 1.0.

The `diffuse` should be set to a suitable texture or color, i.e. white for white bathroom tiles\(^{13}\)

### 2.5.2.3 Stone Materials

Stone is usually fairly matte, or has reflections that are so blurry they are nearly diffuse. The “powdery” character of stone is simulated with the `diffuse_roughness` parameter - try 0.5 as a starting point. Porous stone such as bricks would have a higher value.

Stone would have a very low `refl_gloss` (lower than 0.25) and one can most likely use `refl_hl_only` to good effect for very good performance. Use a nice stone texture for `diffuse`, some kind of bump map, and perhaps a map that varies the `refl_gloss` value.

The `reflectivity` would be around 0.5-0.6 with `brdf_fresnel` off and `brdf_0_degree_refl` at 0.2 and `brdf_90_degree_refl` at 1.0

### 2.5.2.4 Glass

Glass is a dielectric, so `brdf_fresnel` should definitely be *on*. The IOR of glass is around 1.5. Set `diffuse_weight` to 0.0, `reflectivity` to 1.0 and `transparency` to 1.0. This is enough to create basic, completely clear refractive glass.

If this glass is for a window pane, set `thin_walled` to *on*. If this is a solid glass block, set `thin_walled` to *off* and consider if caustics are necessary or not, and set `do_refractive_caustics` accordingly.

Is the glass frosted? Set `refr_gloss` to a suitable value. Tune the `refr_samples` for good quality or use `refr_interpolate`) for performance.

\(^{13}\)
2.5.2.5 Colored Glass

For clear glass the tips in the previous section work. But colored glass is a slightly different story.

Many shaders set the transparency at the surface of the glass. And indeed this is what happens if one simply sets a \texttt{refr\_color} to some value, e.g. blue. For glass done with \texttt{thin\_walled} turned \textit{on} this works perfectly. But for solid glass objects this is not an accurate representation of reality.

Study the following example. It contains two glass blocks of very different size and a sphere with a spherical hole inside of it\textsuperscript{14} plus a glass horse.

\begin{center}
\includegraphics[width=\textwidth]{example.png}
\end{center}

\textit{With a blue \texttt{refr\_color}: Glass with color changes at the surface}

The problems are evident:

- The two glass blocks are of completely different thickness, yet they are exactly the same level of blue.
- The inner sphere is \textit{darker} than the outer.

Why does this happen?

Consider a light ray that enters a glass object. If the color is “at the surface”, the ray will be colored somewhat as it enters the object, retain this color through the object, and receive a second coloration (attenuation) when it exits the object:

\textsuperscript{14}Created by inserting a second sphere with the normals flipped inside the outer sphere. Don’t forget to flip normals of such surfaces or they will not render correctly!
In the illustration above the ray enters from the left, and at the entry surface it drops in level and gets slightly darker (bottom of graph schematically illustrates the level). It retains this color throughout the travel through the medium and drops in level again at the exit surface.

For simple glass objects this is quite sufficient. For any glass using thin_walled it is by definition the correct thing to do, but for any complex solid it is not. It is especially wrong for negative spaces inside the glass (like the sphere in our example) because the light rays have to travel through four surfaces instead of two (getting two extra steps of “attenuation at the surface”)

In real colored glass, light travels through the medium and is attenuated “as it goes”. In the mia_material this is accomplished by enabling the refr_falloff_dist and use the refr_falloff_color and setting the refr_color to white. This is the result:
Glass with color changes within the medium

The above result is clearly much more satisfactory; the thick glass block is much deeper blue than the thin one, and the hollow sphere now looks correct. In diagram form it looks as follows:

\[ d = \text{refr}_\text{falloff}_\text{dist} \text{ where attenuation is } \text{refr}_\text{falloff}_\text{color} \]

The ray enters the medium and during it’s entire travel it is attenuated. The strength of the attenuation is such that precisely at the \text{refr}_\text{falloff}_\text{dist} (d in the figure) the attenuation will match that of \text{refr}_\text{falloff}_\text{color} (i.e. at this depth the attenuation is the same as was received immediately at the surface with the previous model). The falloff is exponential such that at double \text{refr}_\text{falloff}_\text{dist} the effect is that of \text{refr}_\text{falloff}_\text{color} squared, and so on.
There is one minor trade off:

To correctly render the *shadows* of a material using this method one must either use caustics *or* make sure *mental ray* is rendering shadows in “segment” shadow mode.

Using caustics naturally gives the most correct looking shadows (the above image was not rendered with caustics), but will require that one has caustic photons enabled and a physical light source that shoots caustic photons.

On the other hand, the *mental ray* “segment” shadows have a slightly lower performance than the more widely used “simple” shadow mode. But if it is not used, there shadow intensity will not take the attenuation through the media into account properly\textsuperscript{15}.

### 2.5.2.6 Water and Liquids

Water, like glass, is a *dielectric* with the IOR of 1.33. Hence, the same principles as for glass (above) applies for solid bodies of water which truly need to refract things... for example water running out of a tap. Colored beverages use the same principles as colored glass, etc.

![Water into Wine](image)

*Water into Wine*

To create a beverage in a container as in the image above, it is important to understand how the *mia_material* handles refraction through multiple surfaces vs. how the “real world” tackles the same issue.

What is important for refraction is how the transition from one medium to another with a different IOR. Such a transition is known as an *interface*.

\textsuperscript{15}But it could potentially still look “nice”.
For lemonade in a glass, imagine a ray of light travelling through the air (IOR = 1.0) enter the glass, and is refracted by the IOR of the glass (1.5). After travelling through the glass the ray leaves the glass and enters the liquid, i.e. it passes an interface from one medium of IOR 1.5 to another medium of IOR 1.33.

One way to model this in computer graphics is to make the glass one separate closed surface, with the normals pointing towards the inside of the glass and an IOR of 1.5, and a second, closed surface for the beverage, with the normals pointing inwards and an IOR of 1.33, and leaving a small “air gap” between the container and the liquid.

While this “works”, there is one problem with this approach: When light goes from a higher IOR to a lower there is a chance of an effect known as “Total Internal Reflection” (TIR). This is the effect one sees when diving in a swimming pool and looking up - the objects above the surface can only be seen in a small circle straight above, anything below a certain angle only shows a reflection of the pool and things below the surface. The larger the difference in the IOR of the two media, the larger is the chance of TIR.

So in our example, as the ray goes from glass (IOR=1.5) to air, there is a large chance of TIR. But in reality the ray would move from a medium of IOR=1.5 to one of IOR=1.33, which is a much smaller step with a much smaller chance of TIR. This will look different:

![Correct refraction (left) vs. the “air gap” method (right)](image)

The result on the left is the correct result, but how it is obtained?

The solution to the problem is to rethink the modelling, and not think in terms of media, but in terms of interfaces. In our example, we have three different interfaces, where we can consider the IOR as the ratio between the IOR’s of the outside and inside media:

- Air-Glass interface (IOR = 1.5/1.0 = 1.5)
- Air-Liquid interface (IOR = 1.33/1.0 = 1.33)
• Glass-Liquid interface (IOR=1.33/1.5=0.8)

It is evident that in the most common case of an interface with air, the IOR to use is the IOR of the media (since the IOR of air is 1.0), whereas in an interface between two different media, the situation is different.

To correctly model this scenario, we then need three surfaces, each with a separate mia_material applied:

The three interfaces for a liquid in a glass

• The Air-glass surface (blue), with normals pointing out of the glass, covering the area where air directly touches the glass, having an IOR of 1.5
• The Air-liquid surface (green), with normals pointing out of the liquid, covering the area where air directly touches the liquid, having an IOR of 1.33
• The Glass-liquid surface (red), with normals pointing out of the liquid, covering the area where the glass touches the liquid, having an IOR of 0.8

By setting a suitable refr_falloff_dist and refr_falloff_color for the two liquid materials (to get a colored liquid), the image on the left in the comparison above is the result.

2.5.2.7 The Ocean and Water Surfaces

A water surface is a slightly different matter than a visibly transparent liquid.

The ocean isn’t blue - it is reflective. Not much of the light that goes down under the surface of the ocean gets anywhere of interest. A little bit of it is scattered back up again doing a little bit of very literal “sub surface scattering”.

To make an ocean surface with the mia_material do the following steps:
Set `diffuse_weight` to 0.0, `reflectivity` to 1.0 and `transparency` to 0.0 (yes, we do not use refraction at all!).

Set the `refr_ior` to 1.33 and `brdf_fresnel` to `on`. Apply some interesting wobbly shader to `bump` and our ocean is basically done!

This ocean has _only_ reflections guided by the IOR. But this might work fine - try it. Just make sure there is something there for it to reflect! Add a sky map, objects, or a just a blue gradient background. There must be _something_ or it will be completely black.

![A helicopter above the ocean](image)

*The Ocean isn’t blue - the sky is*

For a more “tropical” look, try setting `diffuse` to some slight greenish/blueish color, set the `diffuse_weight` to some fairly low number (0.1) and check the `no_diffuse_bump` checkbox.

Now we have a “base color” in the water which emulates the little bit of scattering occurring in the top level of the ocean.
2.5.2.8 Metals

Metals are very reflective, which means they need something to reflect. The best looking metals come from having a true HDRI environment, either from a spherically mapped HDRI photo\(^{16}\), or something like the mental ray physical sky.

To set up classic chrome, turn `brdf_fresnel` off, set `reflectivity` to 1.0, `brdf_0_degree_refl` to 0.9 and `brdf_90_degree_refl` to 1.0. Set `diffuse` to white and check the `refl_is_metal` checkbox.

This creates an almost completely reflective material. Tweak the `refl_gloss` parameter for various levels of blurry reflections to taste. Also consider using the “round corners” effect, which tend to work very well on metallic objects.

Metals also influence the color of their reflections. Since we enabled `refl_is_metal` this is already happening; try setting the `diffuse` to a “gold” color to create gold.

Try various levels of `refl_gloss` (with the help of `refl_interpolate`) for performance, when necessary.

One can also change the `reflectivity` which has a slightly different meaning when `refl_is_metal` is enabled; it blends between the reflections (colored by the `diffuse`) and normal diffuse shading. This allows a “blend” between the glossy reflections and the diffuse shading, both driven by the same color. For example, an aluminum material would need a bit of diffuse blended in, whereas chrome would not.

\(^{16}\)Many HDRI images are available online.
2.5.2.9 Brushed Metals

Brushed metal is an interesting special case of metals. In some cases, creating a brushed metal only takes turning down the `refl.gloss` to a level where one receives a “very blurred” reflection. This is sufficient when the brushing direction is random or the brushes are too small to be visible even as an aggregate effect.

For materials that have a clear `brushing direction` and/or where the actual brush strokes are `visible`, creating a convincing look is a slightly more involved process.

The tiny grooves in a brushed metal all work together to cause `anisotropic` reflections. This can be illustrated by the following schematic, which simulates the brush grooves by actually modelling many tiny adjacent cylinders, shaded with a simple Phong shader:
As one can see, the specular highlight in each of the cylinders work together to create an aggregate effect which is the *anisotropic highlight*.

Also note that the highlight isn’t continuous, it is actually broken up in small adjacent segments. I.e. the main visual cues that a material is “brushed metal” are:

- Anisotropic highlights that stretch out in a direction *perpendicular* to the brushing direction.
- A discontinuous highlight with “breaks” in the brushing direction.

Many attempts to simulate brushed metals have only looked at the first effect, the anisotropy. Another common mistake is to think that the highlight stretches *in* the brushing direction. Neither is true.

Hence, to simulate brushed metals, we need to simulate these two visual cues. The first one is simple; use **anisotropy** and **anisotropy_rotation** to make anisotropic highlights. The second can be done in several ways:

- With a **bump** map
- With a map that varies the **anisotropy** or **refl_gloss**
- With a map that varies the **refl_color**
Each have advantages and disadvantages, but the one we will try here is the last one. The reason for choosing this method is that it works well together with interpolation.

1. Create a map for the “brush streaks”. There are many ways to do this, either by painting a map in a paint program, or by using a “Noise” map that has been stretched heavily in one direction.

2. The map should vary between middle-gray and white. Apply this map to the `refl_color` in a scale suitable for the brushing.

3. Set `diffuse` to `white` (or the color of the metal) but set `diffuse_weight` to 0.0 (or a small value).

4. Make sure `refl_is_metal` is enabled.

5. Set `refl_gloss` to 0.75.

6. Set `anisotropy` to 0.1 or similar. Use `anisotropy_rotation` to align the highlight properly with the map. If necessary use `anisotropy_channel` to base it on the same texture space as the map.

![Brushed Metal](image)
Chapter 3

Sun and Sky

3.1 Introduction

The mental ray physical sun & sky shaders are designed to enable physically plausible daylight simulations and very accurate renderings of daylight scenarios.

The mia_physicalsun and mia_physicalsky are intended to be used together, with the mia_physicalsun shader applied to a directional light that represents the sun light, and the mia_physicalsky shader used as the scene’s camera environment shader. The environment shader should be used to illuminate the scene with the help of final gathering (which must be enabled) and bounced light from the sun can be handled either by final gather diffuse bounces, or via GI (photons).

3.2 Units

The sun and sky work in true photometric units, but the output can be converted to something else with the rgb_unit_conversion parameter. If it is set to 1 1 1, both the values returned by the mental ray shader API functions mi_sample_light (for the sunlight) and mi_compute_avg_radiance (for the skylight), when sent through the mi_luminance function, can be considered (will numerically match) photometric values in lux.

Since the intensity of the sun outside the atmosphere is calibrated to 127500 lux, this is very bright when seen compared to a more “classical” rendering where light intensities generally range from 0 to 1. The rgb_unit_conversion parameter is applied as a multiplier and should be set to a value below 1.0 (e.g. 0.001 0.001 0.001) to convert the raw lux value to something more manageable.

For convenience, the special rgb_unit_conversion value of 0 0 0 is internally set so that 80000 lux (approximately the amount of light on a sunny day) equals the classical light level
of 1.0.

3.3 Important note on fast SSS and Sun&Sky

To use the *mental ray* fast SSS shaders together with the high dynamic range indirect sun and skylight, it is very important to turn *on* the *Indirect* parameter so the SSS shader can scatter the skylight (which is considered indirect).

It is equally important to turn *off* the *Screen composit* parameter (otherwise the output of the SSS shaders are clamped to a low dynamic range and will appear to render black).

3.4 Common parameters

Some parameters exist both in the *mia_physicalsun* and *mia_physicalsky* and all do the same thing. For physical correctness, it is necessary to keep these parameter *in sync* with each other in both the sun and sky. For example, a sun with a different *haze* value than the sky cannot be guaranteed to be physically plausible.
The most important common parameters are those that drive the entire shading- and colorization model:

- **haze** sets the amount of haze in the air. The range is from 0 (a completely clear day) to 15 (extremely overcast, or sandstorm in Sahara). The haze influences the intensity and color of the sky and horizon, intensity and color of sunlight, softness of the sun’s shadows, softness of the glow around the sun, and the strength of the aerial perspective.

- **redblueshift** gives artistic control over the “redness” of the light. The default value of 0.0 is the physically correct value\(^1\), but can be changed with this parameter which ranges from -1.0 (extremely blue) to 1.0 (extremely red).

\(^1\)Calculated for a 6500K whitepoint.
• **saturation** is also an artistic control, where 1.0 is the physically calculated saturation level. The parameter ranges from 0.0 (black and white) to 2.0 (extremely boosted saturation)

## 3.5 Sun parameters

The **mia_physicalsun** is responsible for the color and intensity of the sunlight, as well as emitting photons from the sun. The shader should be applied as *light shader* and *photon emission shader* on a *directional light source* (it does not work on any other light type).

```plaintext
declare shader "mia_physicalsun" (
    boolean "on" default on,
    scalar "multiplier" default 1.0,
    color "rgb_unit_conversion" default 0.0001 0.0001 0.0001,
    scalar "haze" default 0.0,
    scalar "redblueshift" default 0.0,
    scalar "saturation" default 1.0,
    scalar "horizon_height" default 0.0,
    scalar "shadow_softness" default 1.0,
    integer "samples" default 8,
    vector "photon_bbox_min",
    vector "photon_bbox_max",
    boolean "automatic_photon_energy",
    boolean "y_is_up"
)

version 5
apply light

end declare
```

As mentioned above, the **mia_physicalsun** contains several of the common parameters that are exposed in the **mia_physicalsky** as well (haze, redblueshift etc.). The value of these parameters for the **mia_physicalsun** should match those in the **mia_physicalsky**.

The parameters specific to the **mia_physicalsun** are as follows:

• **samples** is the number of shadow samples for the soft shadows. If it is set to 0, no soft shadows are generated.

• **shadow_softness** is the softness for the soft shadows. A value of 1.0 is the value which matches the softness of real solar shadow most accurately. Lower values makes the shadows *sharper* and higher *softer*.

When **photon_bbox_min** and **photon_bbox_max** are left set to 0,0,0 the photon bounding box will be calculated automatically by the shader. If these are set, they define
a bounding box in the coordinate system of the light within which photons are aimed. This can be used to “focus” GI photons on a particular area-of-interest. For example, if one has modelled a huge city as a backdrop, but is only rendering the interior of a room, mental ray will by default shoot photons over the entire city and maybe only a few will find their way into the room. With the photon_bbox_max and photon_bbox_min parameters one can focus the photon emission of the mia_physicalsun to only aim at the window in question, greatly speeding up and enhancing the quality of the interior rendering.

**automatic Photon_energy** enables automatic photon energy calculation. When this is on, the light source does not need to have a valid energy value that matches that of the sun (it does, however, need a nonzero energy value or photon emission is disabled by mental ray). The correct energy and color of the photons will be automatically calculated. If this parameter is off, the photons will have the energy defined by the lights energy value.

### 3.6 Sky parameters

The mia_physicalsky shader is responsible for creating the color gradient that represent the atmospheric skydome, which is then used to light the scene with the help of final gathering. The mia_physicalsky, when used as an environment shader, also show the sky to the camera and in reflections.

mia_physicalsky also creates a virtual ground plane that exists “below” the model. This makes it unnecessary to actually model geometry all the way to the horizon line - the virtual ground plane provides both the visuals and the bounce-light from such ground.

```plaintext
declare shader "mia_physicalsky" (
  boolean "on" default on,
  scalar "multiplier" default 1.0,
  color "rgb_unit_conversion" default 0.0001 0.0001 0.0001,
  scalar "haze" default 0.0,
  scalar "redblueshift" default 0.0,
  scalar "saturation" default 1.0,
  scalar "horizon_height" default 0.0,
  scalar "horizon_blur" default 0.1,
  color "ground_color" default 0.2 0.2 0.2,
  color "night_color" default 0 0 0,
  vector "sun_direction",
  light "sun",
  # The following parameters are only useful
  # when the shader is used as environment
  scalar "sun_disk_intensity" default 1.0,
  scalar "sun_disk_scale" default 4.0,
  scalar "sun_glow_intensity" default 1.0,
```


boolean "use_background",
shader "background",

# For the lens/volume shader mode
scalar "visibility_distance",

boolean "y_is_up",
integer "flags"
)

version 4
apply environment, texture, lens, volume

end declare

• **on** turns the shader on or off. The default is *on*.

• **multiplier** is a scalar multiplier for the light output. The default is 1.0.

• **rgb_unit_conversion** allows setting the units, described in more detail above. The special value of 0 0 0 matches 80000 lux (light level on a sunny day) to the output value 1, suitable for low dynamic range rendering.

• **horizon_height** sets the “level” of the horizon. The default value of 0.0 puts the horizon at standard “height”. But since the horizon is *infinitely* far away this can cause trouble joining up with any *finite* geometry that is supposed to represent the ground. It can also cause issues rendering locations that are supposed to be at a high altitude, like mountain tops or the top of New York skyscrapers where the horizon *really is* visibly “below” the viewer.

This parameter allows tuning the position of the horizon. Note that the horizon doesn’t actually exist at any specific “height” in 3D space - it is a shading effect for rays that go below a certain *angle*. This parameter tweaks this angle. The total range available range is somewhat extreme, reaching from -10.0 (the horizon is “straight down”) to 10.0 (the horizon is at the zenith)! In practice, only much smaller values are actually useful, like for example -0.2 to push the horizon down just below the edge of a finite visible ground plane.

Note: The **horizon_height** affects not only the visual representation of the horizon in the mia_physicalsky shader, but also the color of the mia_physicalsun itself, i.e. the point where the sun “sets” will indeed change for a nonzero horizon_height.

• **horizon_blur** sets the “blurriness” with which the horizon is rendered. At 0.0 the horizon is completely sharp. Generally low values (lower than 0.5) are used, but the full range is up to 10.0 for a horizon which only consists of blur and no actual horizon at all.
3.6 Sky parameters

- **horizon_height** = 0.0, **horizon_blur** = 0.0
- **horizon_height** = -0.3, **horizon_blur** = 0.2

- **ground_color** is the color of the virtual ground plane. Note that this is a diffuse reflectance value (i.e. albedo). The ground will appear as if it was a lambertian reflector with this diffuse color, lit by the sun and sky only, does not receive any shadows.

- **night_color** is the minimum color of the sky - the sky will never become darker than this value. It can be useful for adding things like moon, stars, high altitude cirrus clouds that are lit long after sunset etc. As the sun sets and the sky darkens, the contribution from **night_color** is unaffected and remains as the “base light level”.

- **sun_direction** is the direction of the sun disk when specified manually. If the **sun** parameter is used, this parameter is ignored.

- **sun** is the way to automatically set the sun direction. It should be the tag of the light instance that contains the directional light that represents the sun - i.e. the same light that has the **mia_physicalsun** shader. This will make the visible sun disk automatically follow the direction of the actual sunlight.

- **Aerial Perspective** is a term used by painters to convey how distant objects are perceived as hazier and tinted towards the blue end of the spectrum. **mia_physicalsky** emulates this with the **visibility_distance** parameter. When nonzero, it defines the “10% distance”, i.e. the distance at which approximately 10% of haze is visible at a **haze** level of 0.0.
To use this effect, the shader must be applied as either a lens or camera volume shader.

- **y_is_up** defines what constitutes “up”. Some OEM integrations of mental ray consider the Z axis “up” and hence this parameter should be off - others consider the Z axis “up” and in that case this parameter should be on.

- **flags** is for future expansion, testing and internal algorithm control. Should be set to zero.

It is important to note that the mia.physicalsky shader treats rays differently. Direct rays from the camera, as well as reflection and refraction rays see the “entire” effect, including the “sun disk” described below. But since the lighting already has a direct light that represents the sun (using the mia.physicalsun shader) the sun disk is not visible to the finalgather rays\(^2\).

These parameters do not affect the final gathering result, only the “visible” result, i.e. what the camera sees and what is seen in reflection and refraction:

- **sun_intensity** and **glow_intensity** is the intensity of the visible sun disk and it’s glow, which can be used to tune the “look” of the sun.

\[
glow\_intensity=5 \quad \text{and} \quad glow\_intensity=0.1
\]

- **sun_scale** sets the size of the visible sun disk. The value 1.0 is the “physically correct” size, but due to the fact that people tend to misjudge the proper size of the sun in photographs, the default is the slightly more visually pleasing 4.0

\[
sun\_scale=1 \quad \text{and} \quad sun\_scale=4
\]

\(^2\)This would otherwise cause noise in the final gathering solution and too much light added to the scene
• When `use_background` is enabled but no `background` has been set, the background of the rendering will be transparent black, i.e. suitable for external compositing. If a `background` shader is supplied, the background of the rendering will come from `that` shader (for example a texture shader that looks up a background photograph of a real location or similar). In either case the `mia_physicalsky` will still be visible in refractions and reflections.
Chapter 4

Utility shaders

4.1 Round corners

CG has a tendency to look “unrealistic” because edges of objects are geometrically sharp, whereas all edges in the real world are slightly rounded, chamfered, worn or filleted in some manner. This rounded edge tends to “catch the light” and create highlights that make edges more visually appealing.

The mia_roundcorners shader can create an illusion of “rounded edges” at render time. This feature is primarily intended to speed up modelling, where things like a table top need not be created with actual filleted or chamfered edges.

![No round corners](image1.png) ![Round corners](image2.png)

The shader perturbs the normal vector, and should be applied where bump maps are normally used, e.g. in the bump parameter if the mia_material.

The function is not a displacement, it is merely a shading effect (like bump mapping) and is best suited for straight edges and simple geometry, not advanced highly curved geometry.
declare shader "mia_roundcorners" (  
  scalar "radius",  
  boolean "allow_different_materials",  
  shader "bump"  
)  
version 2  
apply texture  
end declare

The **radius** parameter defines the radius of the rounding effect, in world space units.

When **allow_different_materials** is **off**, the rounding effect happens only against faces with the same material. If it is **on**, the rounding effect happens against any face of any material.

The **bump** parameter is a passthrough to any other bump shader that handles additional bumping of the surface, for example **mib_bump_map2** or similar.

### 4.2 Tone mapping / Exposure

When rendering physical light levels one runs into the problem of managing the HDRI output of the real physics vs. the limited dynamic range of computer displays. This was discussed in more detail on page 4.

There are numerous shaders and algorithms for doing “tone mapping” (this is a very active area of research within the CG industry), and this shader is a very simple such shader that simply adds a knee compression to “squash” overbrights into a manageable range.

The shader can be applied either as a lens shader (which will tone map the image “on the fly” as it is being rendered) or as an output shader (will tone map the image as a post process). Since this tone mapper affects each pixel individually\(^1\), the former method (as lens shader) is encouraged, since it applies on the sample level rather than the pixel level.

\[^1\text{Many advanced tone mappers weight different areas of the image against other areas, to mimic the way the human visual system operates. These tone mappers need the entire image before they can “do their job”}\]
The operation of this tone mapper is very simple. It takes the high dynamic range color and perform these operations in order:

- **pedestal** is *added* to the color.
- The color components are then multiplied with **gain**.
- The resulting colors are checked if they are above the **knee** value.
- If they are, they are “squashed” by the compression ratio **compression**
- Finally, gamma correction with **gamma** is performed.

That’s the theory. What is the practical use of these parameters?

Changing **pedestal** equates to tweaking the “black level”. A positive value will add some light so even the blackest black will become slightly gray. A negative value will subtract some light and allows “crushing the blacks” for a more contrasty artistic effect.

**gain** is the “brightness knob”. This is the main point where the high dynamic range values are converted to low dynamic range values. For example: if one knows the approximate range of color intensities goes between 0 and 10, this value should then be approximately 0.1 to get this range into the desired 0-1 range.

However, the whole point of tone mapping is *not* to blindly linearly scale the range down. Simply setting it to 0.1 most likely yeilds a dark and boring image. A much more likely value is 0.15 or even 0.2. But a value of 0.2 will map our 0-to-10 range to 0 to 2... what to do about that stuff above 1.0?

That’s where the compression comes in. The **knee** level is the point where the overbrights begin to be “squashed”. Since this is applied *after* the gain, it should be in the range of 0.0 to 1.0. A good useful range is 0.5 to 0.75.

Assume we set it to 0.75. This means any color that (after having **pedestal** added and multiplied by **gain**) that comes out above 0.75 will be “compressed”. If **compression** is 0.0 there is no compression. At a **compression** value of 5.0 the squashing is fairly strong.

Finally, the resulting “squashed” color is gamma-corrected for the output device (computer screen etc.)

The **use_preview** and **preview** parameters are used to make the process of tweaking the tone mapper a little bit more “interactive”. Do the following:

- Disable the **mia_exposure_simple** shader.
- Render the image to a file in some form of HDR capable format (like .exr, .hdr or similar), for example **preview.exr**.
- Enable **mia_exposure_simple** shader again.
• Set the **preview** parameter to the file saved above, e.g. *preview.exr*.

• Enable the **use_preview** parameter.

• Disable any photon mapping or final gathering.

• Re-render. The rendering will be near instant, because no actual rendering occurs at all; the image is read from *preview.exr* and immediately tone mapped to screen.

• Tweak parameters and re-render again, until satisfied.

• Re-enable any photons or final gathering.

• Turn off **use_preview**.

• Voila - the tone mapper is now tuned.
Chapter 5

Advanced topics

This section is mainly of interest to OEM integrators of mental ray shaders into applications.

5.1 mia_material API

The mia_material exposes features that allows a much deeper integration into OEM applications than ever before. Most notably it exports:

- an interface for obtaining subcomponents of the rendered result (diffuse, reflections, etc.).
- an interface for obtaining separate “diffuse” and “specular” lighting.

This is implemented as a C API for shader developers, utilizing shader states\(^1\) for passing the information in and out of the mia_material.

The keys and structs used are available in the public file mia_material_api.h which is listed in its entirety below.

5.1.1 Obtaining sub-components of the rendering

To obtain independent results from the various shading components one must wrap the mia_material shader in some other shader that sets up the appropriate shader state, calls mia_material and then reads the info stored in the shader state for further processing.

Setting up the shader state structure is the responsibility of this wrapping shader. The structure member struct_size must be initialized to the size of the struct (for version control) and in_use initialized to miFALSE.

---

\(^1\)See the book Programming mental ray for more details.
Then mia_material is called, for example with the help of mi_call_shader_x().

After the call the in_use parameter is inspected. If true, the rest of the structure contains valid values.

A sample shader saving into separate mental ray framebuffers is listed on page 66.

5.1.2 Defining characteristics of light sources

This interface allows light shaders to inform the mia_material if they are emitting specular or diffuse light (or both, which is the default).

This shader state is set up by the mia_material and filled with defaults. Light shaders should only test for the presence of the shader state. If it is missing, the light shader needs to take no further action. But if the shader state exists, the light shader can modify the structures values, i.e. set the affects_diffuse or affects_specular scalars to suitable values.

See listing below for the meaning of each parameter:

5.2 mia_material_api.h

file listing

/*****************************************************************************
* Copyright 1986-2006 by mental images GmbH, Fasanenstr. 81, D-10623 Berlin,
* Germany. All rights reserved.*****************************************************************************
* Created: 06.04.12
* Module: architectural
* Purpose: the mia_material PUBLIC API
* ...
* Exports:
* ...
* mia_material_api_*()
* ...
* History:
* 06.05.10: Created
* 06.05.22: Extended
* ...
* Description:
*****************************************************************************

#define miA_MATERIAL_API_STORAGE "miXMST"

/*
   Protocol for extracting arbitrary outputs from the mia_material:
1. Create a shader state named after the macro miA_MATERIAL_API_STORAGE pointing to a struct mia_material_api_outputs

2. Set its struct_size to sizeof(mia_material_api_outputs);

3. Set in_use to miFALSE; No further initialization is needed.

4. Call the mia_material shader.

5. Upon return of mia_material, see if in_use is miTRUE. If so, the structure in the shaderstate will be filled in with the topmost shaders values.

*/

typedef struct {
    /* Set before calling mia_material shader: */
    miUint struct_size; /* Set to sizeof() struct */
    miBoolean in_use; /* Set to miFALSE */
    /* Return values after calling mia_material shader */
    miScalar opacity; /* scalar opacity */
    miColor indir_result; /* Indirect shading (FG and GI) */
    miColor diff_result; /* Diffuse shading */
    miColor spec_result; /* Specular/Highlights */
    miColor tran_result; /* Translucency */
    miColor refl_result; /* Reflections */
    miColor refr_result; /* Refractions */
    miColor add_result; /* "Additional color" */
    miColor ao_result; /* AO contribution only */
    miColor diff_level; /* Actually used diffuse color/level */
    miColor refl_level; /* Actually used reflection color/level */
    miColor refr_level; /* Actually used refraction color/level */
    miColor tran_level; /* Actually used translucency color/level */
    miRay_type type; /* Ray type for the stored data */
} mia_material_api_storage;

#define miA_MATERIAL_API_LIGHTDATA "miXMLD"

typedef struct {
    /* All below set up by mia_material prior to calling each light */
    miUint struct_size; /* Set to sizeof() struct, for versioning */
    /* Read only's: DO NOT change in light shader */
    miScalar glossiness; /* Light shader can make decisions based on glossiness */
    miScalar importance; /* Light shader can make decisions based on importance */
    /* Modify as needed in the light shader */
    /* How much does this light affect diffuse and specular? */
    miScalar affect_diffuse; /* defaults to 1.0 */
    miScalar affect_specular; /* defaults to 1.0 */
    /* Is this the mr Sun? (should only be set by the mr sun) */
} mia_material_api_lightdata;
miBoolean is_mr_sun; /* defaults to miFALSE */
/* If this is a visible area light, but it still desires to
get a highlight, set force_specular to true */
miBoolean force_specular; /* defaults to miFALSE */
} mia_material_api_lightdata;

5.2.1 Sample shader source

Sample shader (code snippet) for saving the output of a mia_material into frame buffers:

#include <shader.h>
/* Parameter struct */
typedef struct {
    miTag mtl; /* Tag of mia_material shader instance */
} mia_material_out_wrapper;

DLLEXPORT miBoolean mia_material_out_wrapper(
    miColor *result, 
    miState *state, 
    mia_material_out_wrapper *param)
{
    /* Tag of actual mia_material (sent in as parameter to wrapper shader) */
    miTag mtl = *mi_eval_tag(&param->mtl);
    /* Struct to store the data in */
    mia_material_api_storage mtldata;
    /* We also need a pointer */
    mia_material_api_storage *dp;
    /* Need the key len */
    static int klen = sizeof(miA_MATERIAL_API_STORAGE);

    /* Initialize the struct */
    mtldata.struct_size = sizeof(mtldata);
    mtldata.in_use = miFALSE;

    /* Create/set the shader state */
    mi_shaderstate_set(state, miA_MATERIAL_API_STORAGE, &mtldata, sizeof(mtldata), 0);

    /* Call shader */
    mi_call_shader_x(result, miSHADER_MATERIAL, state, mtl, NULL);

    /* Check if the data is there */
    dp = mi_shaderstate_get(state, miA_MATERIAL_API_STORAGE, &klen);

    /* So, was valid data written? */
    if (dp && dp->in_use)
    {
        /* Diffuse to fb #10 */
    }
mi_fb_put(state, 10, &dp->diff_result);

    /* Reflection to fb #11 */
    mi_fb_put(state, 11, &dp->refl_result);

    /* Refraction to fb #12 */
    mi_fb_put(state, 12, &dp->refr_result);

    /* etc. */
}

return miTRUE;
}

Sample light shader (code snippet only) to set a light to “specular only”

static int klen = sizeof(miA_MATERIAL_API_LIGHTDATA);

mia_material_api_lightdata* ld =
    (mia_material_api_lightdata*)mi_shaderstate_get(state, miA_MATERIAL_API_LIGHTDATA, &klen);

    /* Is there a shader state? */
if (ld)
{
    /* Our light is specular-only (no diffuse) */
    ld->affect_diffuse = 0.0f;
    ld->affect_specular = 1.0f;
}