

Occlusion tutorial

Document version 1.0
April 27th, 2005

Copyright Information

Copyright © 1986-2007 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®, mental ray®, mental matter®, mental mill™, mental queue™, mental q™, mental world™, mental map™, mental earth™, mental mesh™, mental™, Reality™, RealityServer®, RealityPlayer®, RealityDesigner®, MetaSL™, Meta™, Meta Shading™, Meta Node™, Phenomenon™, Phenomena™, Phenomenon Creator®, Phenomenon Editor™, Phenomill™, Phenograph™, neuray™, iray®, imatter®, Cybernator™, 3D Cybernator™, Shape-By-Shading®, SPM®, NRM™, and rendering imagination visible™ are trademarks or, in some countries, registered trademarks of mental images GmbH, Berlin, Germany.

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 Occlusion	1
1.1 Intro	1
1.1.1 Further resources	1
1.2 What is Occlusion	1
1.3 Usage of Occlusion	2
1.4 The occlusion shaders set	4
1.4.1 Example uses of the occlusion shaders	6
1.4.1.1 Ambient (or diffuse) occlusion	8
1.4.1.2 Diffuse, environment sampled, occlusion	9
1.4.1.3 Reflective, environment sampled, occlusion	9
1.4.1.4 Bent normals	11
1.5 Using mib_amb_occlusion as a light shader	17
1.6 Ambient Occlusion vs. standard Final Gathering	18
1.6.1 Tips	20
1.7 Ambient Occlusion using Final Gathering	21

Chapter 1

Occlusion

1.1 Intro

The **base** shader library from mental images¹ provides a set of utility shaders helpful to gather and control occlusion information. This document covers the practical usage of such shaders.

Please note that the shaders described in this tutorial require mental ray version 3.4.3 and newer. The library we document here contains 2 shaders introduced in 3.4.3 `mib_fg_occlusion` and `mib_bent_normal_env` and an updated version of `mib_amb_occlusion`.

1.1.1 Further resources

If you need further information, please refer to the mental ray **base** shader library documentation, which is provided in HTML form. This document includes parts of that documentation, and tries to put its contents more in the usage perspective.

For further information about mental ray refer to its online documentation and to the two books by Thomas Driemeyer “Rendering with mental ray” and “Programming mental ray”²

Another useful source of information is <http://www.lamrug.org> LAmrUG, the Los Angeles mental ray User Group.

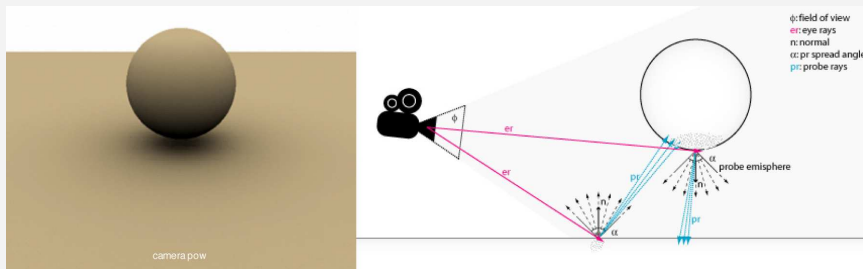
1.2 What is Occlusion

Occlusion is the extent to which the area above a point is covered by other geometry and is generally used as a simplified method to “simulate” Global Illumination. This is achieved by

¹part of mental ray and present in its OEM integrations

²both edited by <http://www.springer.at> SpringerWienNewYork

tracing a number of “probe rays” inside the hemispherical area above the point and testing to what extent this region is blocked. Obviously the raytracing algorithm is required to send such rays. Our implementation is accomplished at the shading level through the `mib_amb_occlusion` shader.



Occlusion through `mib_amb_occlusion`

Using this shader is not the only way to get an occlusion effect in mental ray. In fact, the Final Gathering algorithm can also be used to achieve a similar, though slightly different, and in some senses more “accurate” effect. In mental ray one can also calculate actual Ambient Occlusion by using the Final Gathering algorithm.

At the end of this document you will find a table which compares standard Ambient Occlusion vs. Final Gathering and Ambient Occlusion through Final Gathering.

1.3 Usage of Occlusion

Occlusion does have several uses on the practical side:

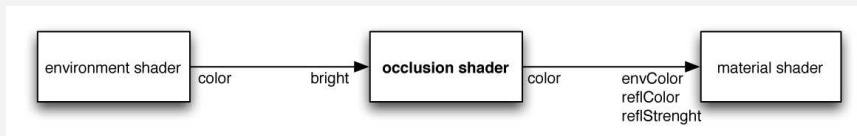
- **Ambient Occlusion** The “classic” Ambient Occlusion effect, also called diffuse occlusion, can be used to scale down the contribution of ambient light (which in turn may come from a diffuse environment map shader). For this use, the shader works well when assigned to the `ambient` parameter of a material like `mib_illum_lambert`. Ideally, Ambient Occlusion should be used with diffuse illumination models such as the lambertian one.



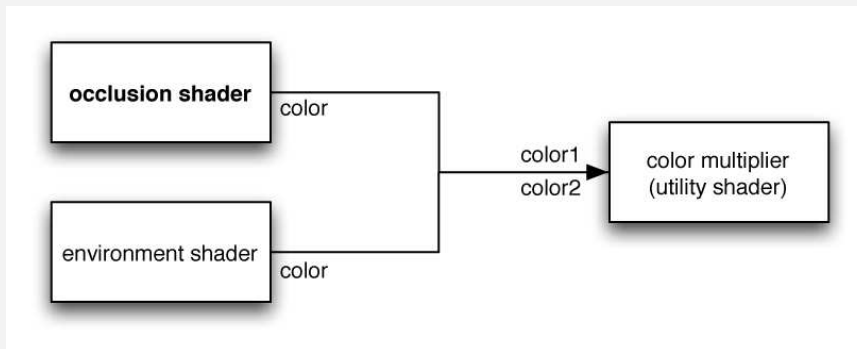
Ambient Occlusion shading graph

- **Reflective Occlusion** A more specific occlusion effect for reflective materials simulating reflections with environment maps, allowing the shader to scale down the contribution

from an environment (reflection) map. For this use, the shader is generally plugged into an environment or reflection shader slot of a material, and the actual environment map image is plugged into the **bright** parameter. An alternative is to allow the return value of the occlusion shader to modulate the strength attribute of a reflection map shader.

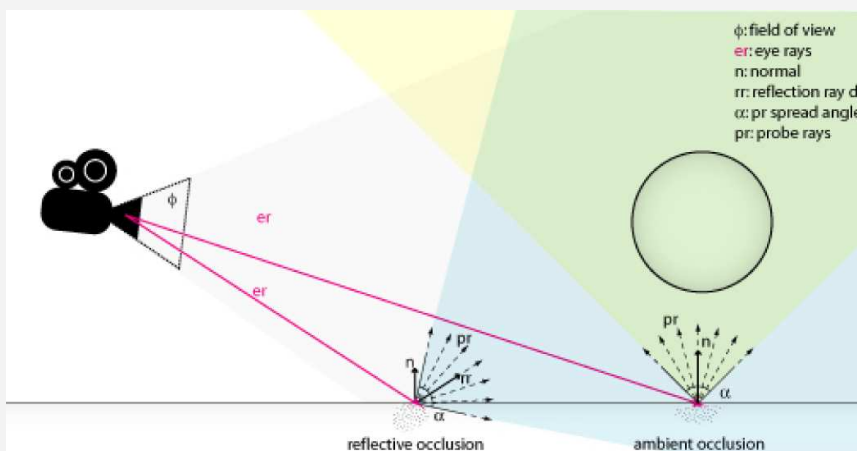


Reflective Occlusion shading graph



Reflective Occlusion second shading graph

Reflective occlusion sends “probe rays” around the reflection ray direction instead of around the geometry normal vector, as it happens in regular Ambient Occlusion. The following diagram shows the difference.



Ambient Occlusion Vs. Reflective Occlusion

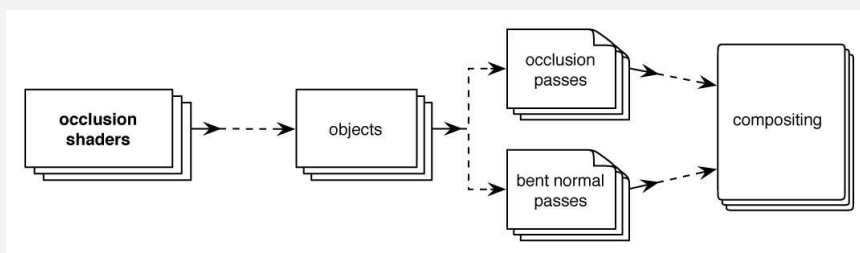
- **Occlusion files for compositing** Often in feature film production one wants the extra control and flexibility in tweaking the lighting situation in post production. (It is much

easier to just change the opacity of a layer in a compositing application and see the result in realtime than changing a shading parameter and re-rendering the entire image... or animation!) For this reason it is actually quite seldom one renders a complete rendering including all the “bells and wistles”. Instead, rendering is broken into separate passes for later compositing.

One common such “pass” is an *occlusion pass*: a separate grayscale rendering of Ambient Occlusion only that is multiplied with an ambient lighting pass and/or a diffuse lighting pass (or similar) in post.

Another is a *bent normal* pass, which encodes the direction vector to look up the ambient environment. This pass actually moves nearly the entire lighting equation to post production, allowing almost complete re- lighting of the objects in the compositing stage! The primary purpose of “bent normals” is to delay the ambient lighting to a later stage. “Later” may be either in an external compositing application (having rendered a bent normal pass and using as a layer in said application) or in a later rendering pass within mental ray (having rendered to a texture, performing the lookup with `mib_bent_normal_env`).

`mib_amb_occlusion` supports creating both these passes **at once** by storing the *bent normal* in the color components, and the *occlusion* in the alpha component of the rendering, by turning on **occlusion_in_alpha** flag. For this to work, the rendering must be saved with neither the alpha nor the color modified from the exact output value of the shader. In standalone mental ray this means the “colorclip” mode must be set to “raw” in the options block. Standalone default is “RGB”.³



The compositing graph logic

Finally the shader may be used as a **light shader**. The light source must be an area light of the **user** type. This generates an ambient light source with built in occlusion, useful if you don’t want to apply the shader on a per-object basis.

Occlusion as light shading graph

1.4 The occlusion shaders set

On the practical side, you will end up using the following shaders, all part of the **base** shader library included with the mental images release of mental ray. These occlusion shaders become available after the library is correctly linked and included.

³The availability, and the default value, of this option varies between OEM integrations of mental ray - consult your application documentation for details.

- **mib_amb_occlusion** This is the occlusion shader itself, which handles all computations. Following here its .mi declaration:

```
color "mib_amb_occlusion" (
    integer "samples"
    color   "bright"
    color   "dark"
    scalar  "spread"
    scalar  "max_distance"
    boolean "reflective"
    integer "output_mode"
    boolean "occlusion_in_alpha"
    scalar  "falloff"
    integer "id_incl excl"
    integer "id_nonself"
)
version 2
```

- **mib_bent_normal_env** This is a shader to be used after light mapping (also known as “texture baking”): it reuses previously rendered occlusion passes allowing environment lighting retouch without recomputing occlusion. Following here its relative .mi declaration:

```
color "mib_bent_normal_env" (
    color      "bent_normals"
    boolean    "occlusion_in_alpha"
    color      "occlusion"
    scalar     "strength"
    shader     "environment",
    integer    "coordinate_space"
    integer    "env_samples"
    scalar     "samples_spread"
    transform  "matrix"
)

```

- **mib_fg_occlusion** This is a utility shader which returns occlusion calculated with the help of the Final Gathering algorithm if the Final Gathering algorithm is active. If Final Gathering is instead 'OFF' it returns a result choosen by the user.⁴ Following here its relative .mi declaration:

```
color "mib_fg_occlusion" (
    color "result_when_fg_is_off"
)
version 1
```

⁴It could return result of the `mib_amb_occlusion` shader as well as the result of a regular illumination model.

1.4.1 Example uses of the occlusion shaders

The setup in use for the following examples is fairly simple, as the helicopter model is the only geometry we have in the scene. There is also an `mib_lookup_spherical` node for environment reflection and environment sampling. The 3D model is kindly provided by Zhang Jian. A `mib_light_infinite` takes care of the directional lighting while the new `mib_light_cie` takes care of the light color in Kelvin degree temperature; finally, raytraced shadows are active.

When baking, the `mib_lightmap_write` is used, the texturing comes from a `mib_texture_vector`, and the lightmap texture is declared as “writable”. When recovering the baked occlusion, the `mib_bent_normal_env` shader is used and textures are looked up with `mib_texture_lookup` in the texture space defined by `mib_texture_vector`.

So this is a quick render without any occlusion effect:



Ambient Occlusion Our helicopter with direct illumination and no occlusion

You can easily see that much of the geometry detail is unrevealed due to the intrinsic nature of direct illumination and its shadowing.

The “traditional” approach to solve this issue would be to add ambient light of a constant level, but this tends to look very unrealistic, since now the details are not lost in black, but instead lost in a constant color, the following rendering shows our copter with a fixed “ambient” color of some dark-ish gray (note that you can either add ambient color via an ambient light or the

ambient color of your shading model):



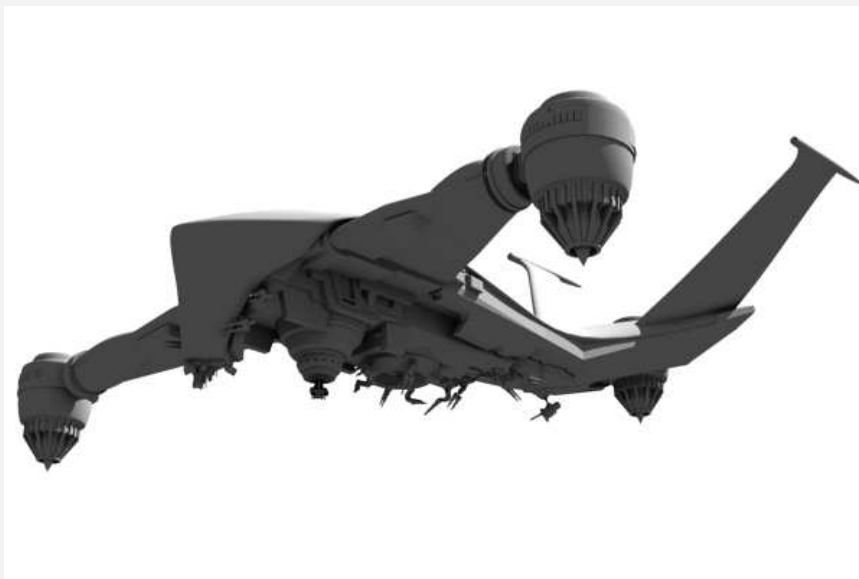
Ambient Occlusion Our helicopter with direct illumination and an ambient light (no occlusion yet)

The “modern” approach is instead using our occlusion shader which provides various occlusion effects available. We can decide which occlusion effect to output by tweaking the `mode` and `reflective` parameters:

- Classic-style Ambient Occlusion (also known as diffuse occlusion)
- Reflective Occlusion
- Environment sampled Occlusion (both diffuse and reflective)
- Bent normals (in world, camera and object space)

Up to now you’ve seen the shading graph logic for all the modes, now let’s analyze them one by one with reference images where the main parameters are pointed out. You can still refer to the example scenes which are provided with this tutorial in `.mi` format.

1.4.1.1 Ambient (or diffuse) occlusion



Ambient occlusion,

output mode = 0

In this case we send probe rays around the geometry normal, adding a percentage of color (in the range defined by the *dark* and *bright* input parameters). As a result, the more occluded the geometry, the darker the color we see, and conversely, the less occluded, the lighter the color. This is the default behavior of the `mib_amb_occlusion` shader, as this is the default setting for the *output_mode*.

You may think of the occlusion effect as a “layer” to composite over a directionally lit, or diffuse, layer. The composite can be performed as desired using additive, multiplied, or other combination methods. Because this is how it is typically used in production, it is important to keep it in mind while setting up a scene that performs the equivalent calculation. Visually speaking, the helicopter image now reveals much more of the detail previously hidden by shadow.

1.4.1.2 Diffuse, environment sampled, occlusion



Diffuse, environment sampled, occlusion, output mode = 1

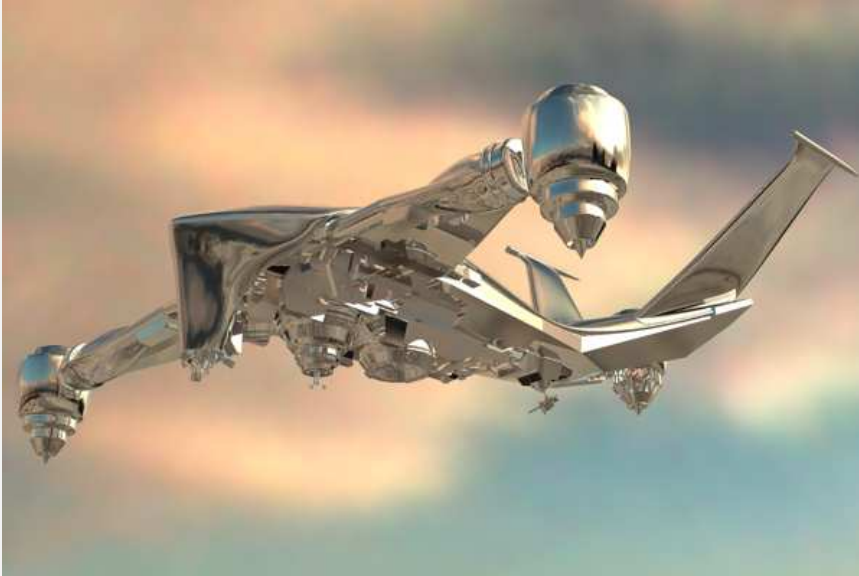
Here we have a sky environment visible (`mib_spherical_lookup` attached to the environment port of your camera); by specifying *output mode = 1*, the shader multiplies a percentage of color in the range between the *dark* and the *bright* color, by the environment color. As above, this can be used as a layer over a diffuse layer. And, we are still calculating occlusion around the geometry normal.

When doing environment sampling you should consider the use of textures with a higher dynamic range, they will return colors in a wider range allowing “super-white” colors which greatly enhance visual realism. The use of such textures, in primis the .EXR format, is quite common in production, if you wish to know more about this you can refer to the mental ray documentation and you can find various resources and tutorials in the internet. Here we are using a free .HDR texture, though the .HDR format is slightly less efficient than the .EXR.

1.4.1.3 Reflective, environment sampled, occlusion

If you apply a reflection map to a model with specular shading, it will show reflections *everywhere* on the object’s surface, even in areas where the object in reality would self-reflect.

This is the old-style approach.



A reference image with a pure chrome-like reflection, no occlusion here

With a reflective occlusion map, the reflection map gets attenuated in these areas where the object's reflection is blocked (by itself or other objects).



Reflective, environment sampled, occlusion: reflective ON, output mode = 1

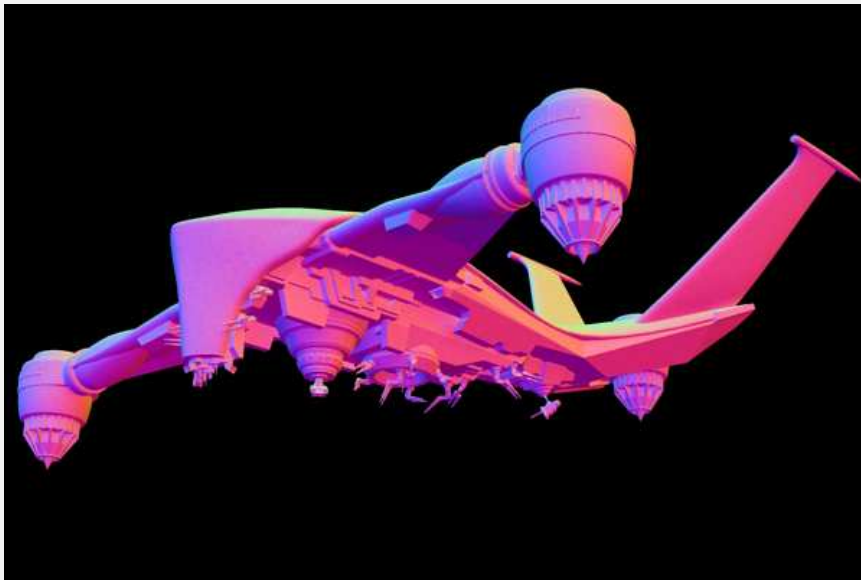
This image features an occlusion effect calculated around the reflective ray direction instead of the direction of the geometry normal vector. This is necessary *especially* when the illumination

model of the occluded object in use is highly specular. Obviously you are not forced to follow this rule and you can use the reflective occlusion also with other illumination models, visually speaking the effect will be slightly more pronounced; keep in mind, that reflective occlusion can enhance the realism of reflection mapping greatly.

When dealing with reflective occlusion it is quite logical to be in environment sampling mode (output mode = 1), but then again you are not forced to do that. In this first case, let's stick to the logical approach and use a visible environment. We are using the same setup as in the previous image, but in extension the `mib_sphericallookup` is plugged into the `bright` parameter of the `mib_ambient_occlusion` shader. In this way, the reflections on the helicopter are coherent with the environment, and we are “layering” on an occlusion effect sampled around the reflective ray direction in the color range between the dark color and the color of the environment.

There is another shading graph layout which you could use for reflective environment sampled occlusion: just use the `mib_color_mix` shader in “mode 4” (color multiplier) and multiply the outcolor of the environment shader with the outcolor of the `mib_amb_occlusion` shader.

1.4.1.4 Bent normals



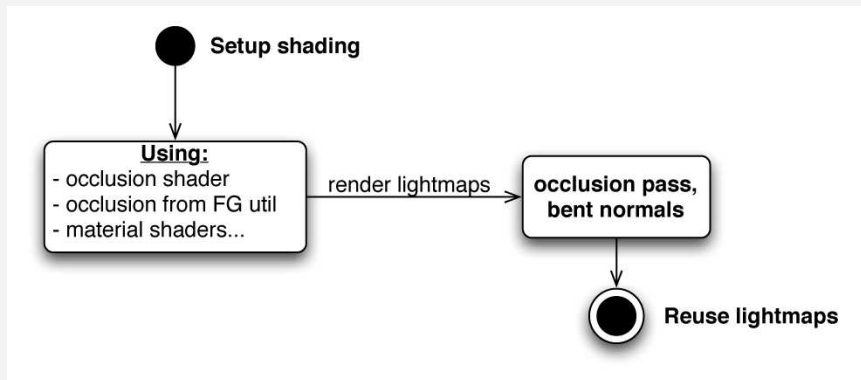
bent normal in world space:

output mode = 2

“Bent normal” is a term used for the average un-occluded direction vector from a surface point. For completely un-occluded surfaces this is the same as the normal vector, but for surfaces occluded by other geometry it points in the direction in which the least amount of occluding geometry is found. Bent normals are used as an acceleration technique for Ambient Occlusion, allowing for very fast rendering that look like Global Illumination or Final Gathering lit by an environment.

Ambient occlusion as done by the `mib_amb_occlusion` shader is a ray tracing technique that casts potentially large numbers of probe rays to determine to which extent a surface point is occluded. The speed of this operation depends on the number of rays (**samples**), the reach of the rays (**max_distance**) and the complexity of the scene.

When rendering an animation (or multiple views of the same scene), any object that does not move, does not change shape, or has no moving occluding object nearby will yield the same result for every frame. Therefore one can “bake” (render to a file) the ambient occlusion solution *once* in a first rendering pass, and re-use this result in subsequent rendering passes for any number of frames, with potentially huge performance gains. If one also “bakes” the average un-occluded direction (the bent normal) to a texture, the entire process of lighting the object based on an environment is moved to this second rendering pass, without having to trace a single ray.

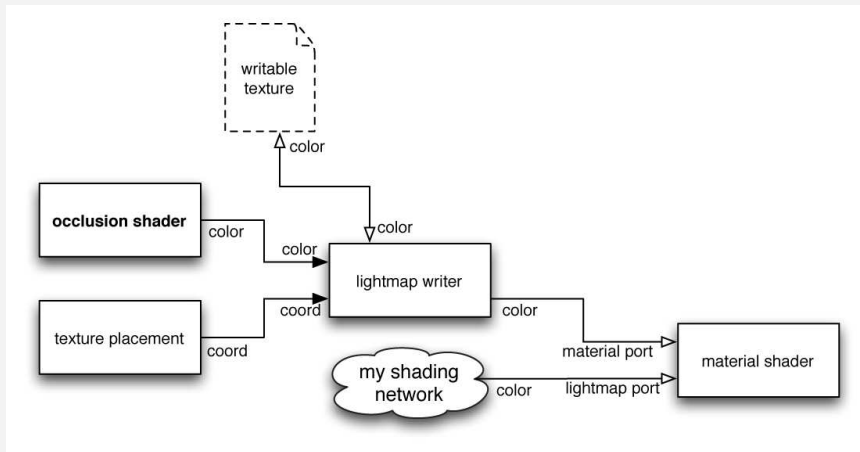


The lightmapping process in a nutshell

By setting the **output_mode** parameter to 2, 3, or 4 on the ambient occlusion shader `mib_amb_occlusion`, bent normals are returned with the vector being encoded as a color where x is red, y is green, and z is blue.

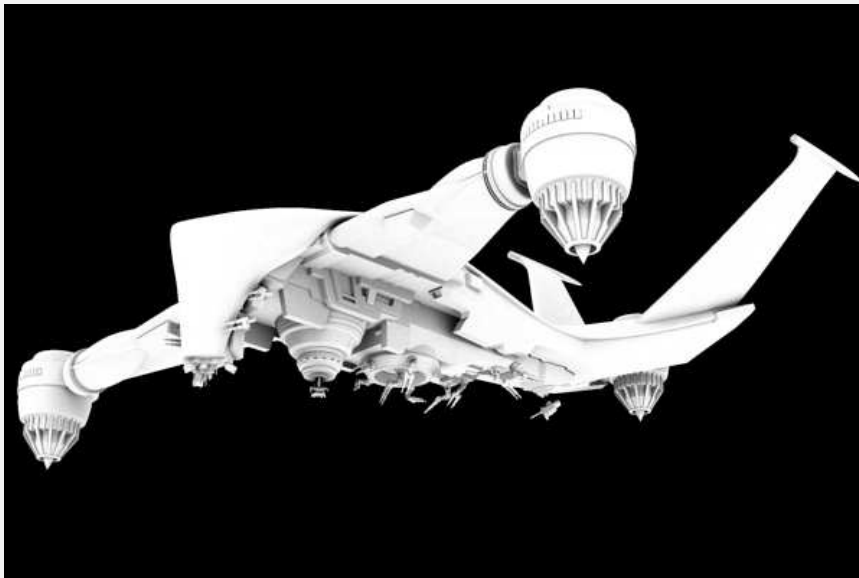
If **occlusion_in_alpha** is enabled, the scalar occlusion value is returned in the alpha channel. This color can be baked into a texture, for example with the help of `mib_lightmap_write` by

putting `mib_amb_occlusion` into its **input** parameter and rendering.



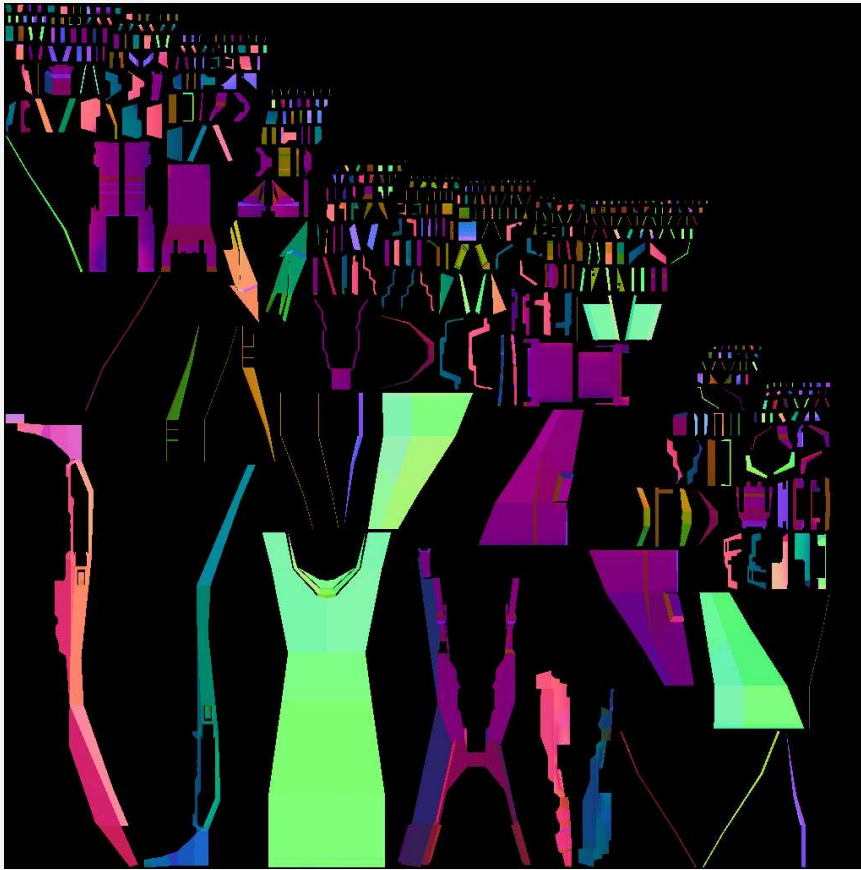
The shading graph for the lightmapping operation

Please note that you can use the `occlusion_in_alpha` with any value of the `output_mode`.



occlusion can be saved in the alpha channel:

occlusion in alpha ON

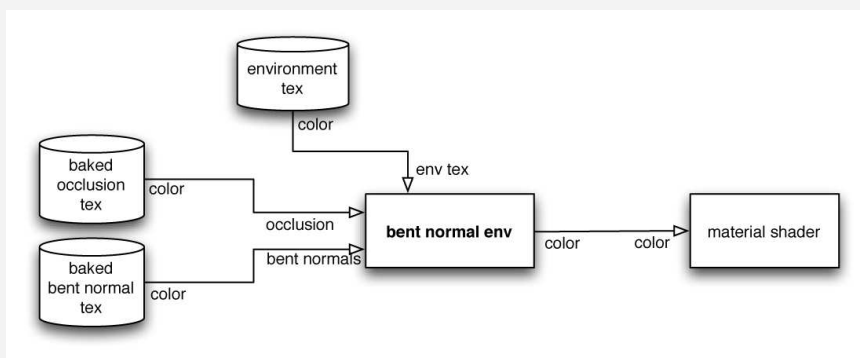


bent normal can be baked with mib_lightmap_write



occlusion can be baked with `mib_lightmap_write` in the alpha channel

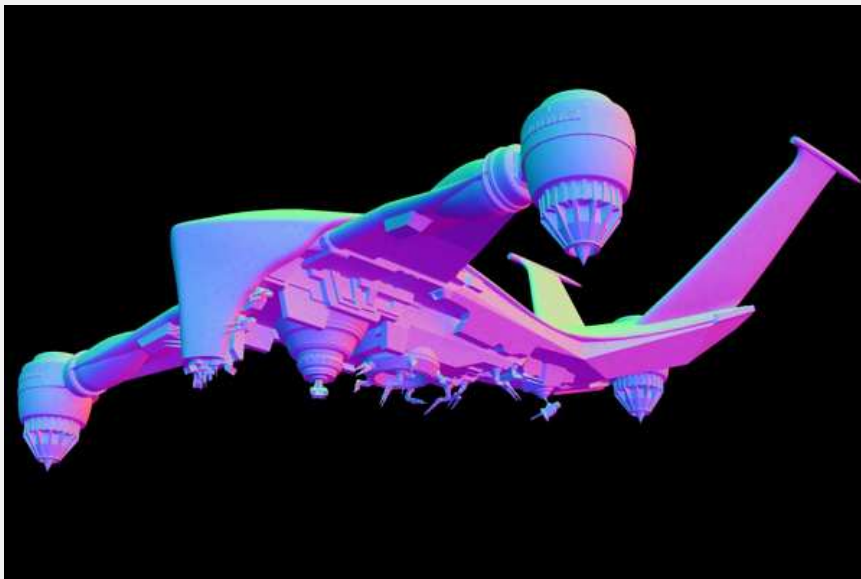
Once the texture file is generated, `mib_bent_normal_env` can be assigned to a surface shader (for example to the **ambient** parameter of `mib_illum_phong`). There, it will look up an environment to light the object based on the baked bent normal texture, which should be connected into the **bent_normals** parameter of `mib_bent_normal_env`.



Using `mib_bent_normal_env` into a shading graph

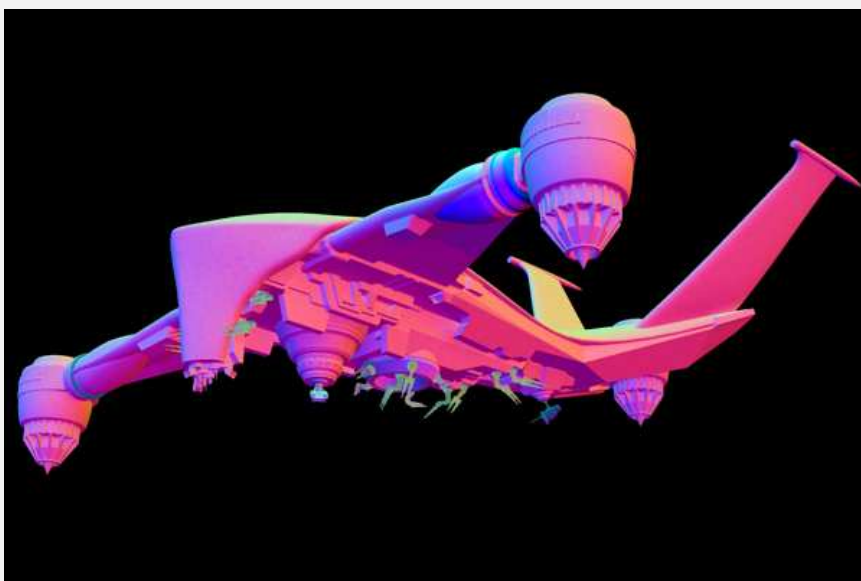
This allows an extremely low-overhead simulation of global illumination-like effects for rigid objects and is especially suitable for animations. The technique does not work on deformable objects since their occlusion, shape, and normals can vary from frame to frame. For deformable objects, it is better to apply the occlusion shader directly, without baking its output.

The visual results when using the shader in mode 3 and 4 are the following:



bent normal in camera space:

output mode = 3



bent normal in object space:

output mode = 4

1.5 Using mib_amb_occlusion as a light shader

This is a trick for expert users: the `mib_amb_occlusion` shader also works as a light shader (you’ve seen previously its shading graph); this results into having an “environment light” with built-in occlusion. When using the shader in such a configuration there is no visual difference as per the Ambient Occlusion effect, but it can be handy in one particular situation (hence why we declare it as a “trick”). The following are the pros and cons:

- Pros:
- The user is relieved from assigning the shader to every surface which can be annoying if your scene is made of thousands of objects and if you have nested scenes.
 - This “environment light” correctly affects multiple materials automatically, hence it is very useful if your material shader does not have an “ambient” parameter, such as for example the `dgs_material`.
- Cons:
- The shader still sends probe rays (not shadow rays) and hence does not respect shadow shaders or surface transparencies; you should use the shader in this way as an “environment light” while relying on another light shader for shadows and transparencies.
 - Lights in mental ray have no specific “diffuse” or “specular” light outputs. This may cause a light using `mib_amb_occlusion` to generate an undesirable specular highlight.⁵

To do this, the light instance in the `.mi` file must be of the `user` type:

```
shader "my_light_shader" "mib_amb_occlusion" (
    "samples"      16,
    "bright"        0.3 0.3 0.3 1,
    "dark"          0 0 0 1,
    "output_mode"  0
)

light "my_light" = "my_light_shader"
    origin 0 0 0
    user 1 1
end light

instance "my_light_instance"
    "my_light"
    transform
        1 0 0 0
        0 1 0 0
        0 0 1 0
        0 0 0 1
    ()
end instance
```

⁵Some OEM application have shaders with specific flags for “diffuse” or “specular” or can solve the issue with custom tricks.

The user area light is activated by the `user` keyword in the light block. Samples is set to 1 because to the lightsource the `mib_amb_occlusion` call is one sample, even though multiple samples happen internally inside the shader.

The light position is not important since this creates a light “coming from all directions at once”, however, depending on the surface shader, it may influence the location of the above mentioned potentially unwanted specular highlight.

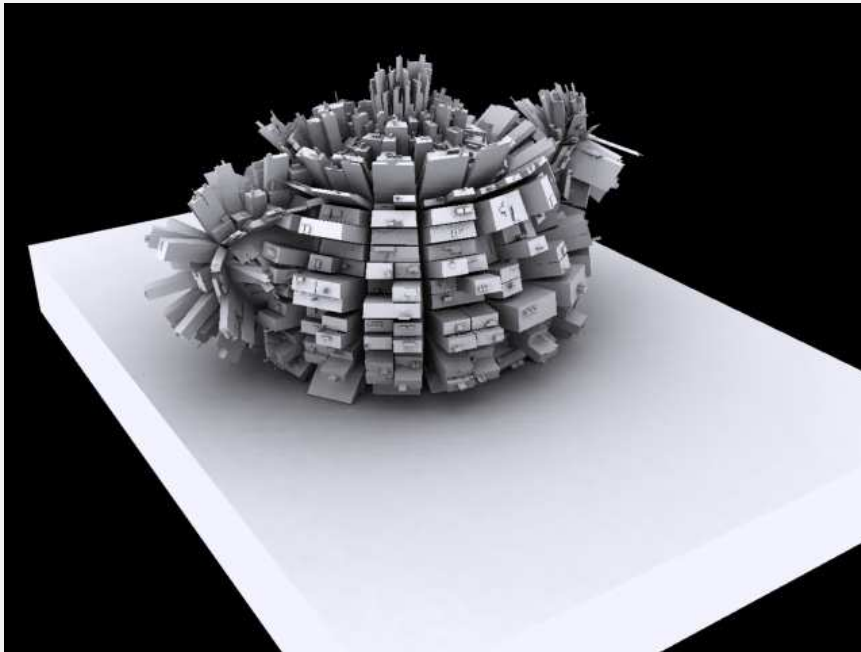
1.6 Ambient Occlusion vs. standard Final Gathering

What is the difference between Ambient Occlusion and Final Gathering?

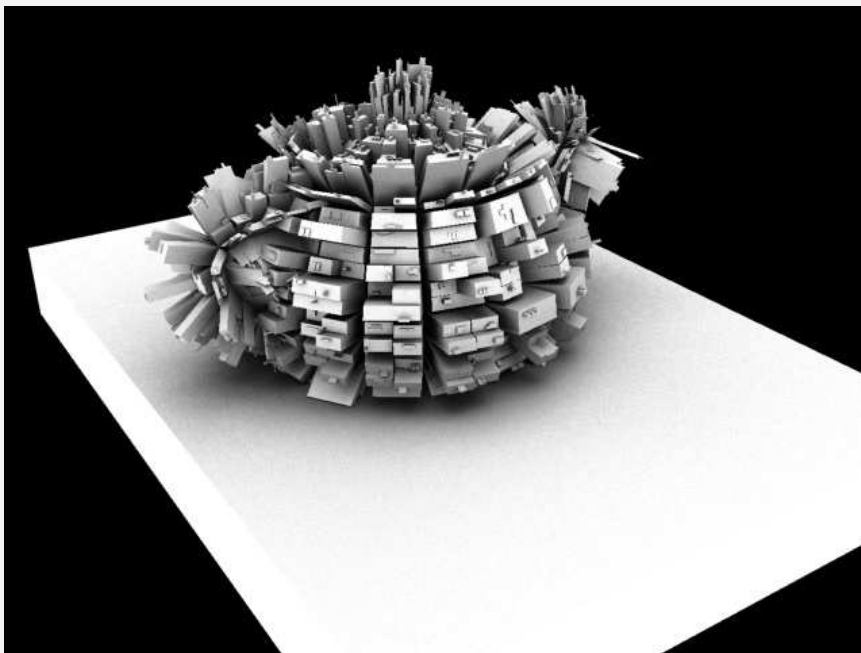
Final Gathering is concerned with the bounced light and multibounce transparency, reflection, refraction and diffuse rays, whereas Ambient Occlusion is more concerned with the *absence* of light, simulating the bounced light with an assumed omnipresent “environment light” (generally coming from a diffuse environment map or a solid color) and letting occluding geometry *block* this light.

With Ambient Occlusion, the rays shoot out from the shading point, whereas, with Final Gathering they shoot out from special final gather points created in a preprocess, and this data is smoothed/filtered to be used later by shaders. The trick with Final Gathering is to *tune* this filtering, to achieve results that are both smooth, yet not over-filtered (a situation which has the potential to cause visible flicker in animations and loss of small details).

The output of `mib_amb_occlusion` is not filtered and hence is never as smooth as Final Gathering, but instead any flicker is never larger than a sample (i.e. less than a pixel, generally). There is no risk in losing small surface details.



Occlusion obtained with regular Final Gathering



Ambient Occlusion calculated via the mib_amb_occlusion

Ambient Occlusion can be simpler to use, and can be computationally cheaper (especially if “baked” to an object). It also creates less “surprises” - all that Ambient Occlusion does is

attenuate a well known contribution of “ambient light from the environment”. This means that every object is given a well-known addition of this “environment light” *or less*. Things can only become *darker* (never brighter) than this level, since no actual bouncing of light occurs.

In contrast, when using Final Gathering, true bouncing of light is performed, and while an environment map can still act as a “well known base level” of light for all objects, the result can be both brighter or darker than this level, due to the bounces. Naturally, in many cases this is desirable or even highly necessary, but in other cases this can cause unwanted problems and difficulty compositing the rendering into a background plate. Since the bouncing itself is managed by the surface shaders of the objects, the Ambient Occlusion effect depends on how well these shaders behave with bouncing light (FG rays). Some shaders may produce odd non-physical results from too much surface-to-surface bounce.

When an Ambient Occlusion probe ray hits a (semi)transparent object, the object is treated as any regular fully opaque object, hence occlusion rays do not traverse semitransparent geometry. However, with Final Gathering, a separate trace depth option handles multibounce transparency, reflections, refraction for rays originated at the final gather point.

With Ambient Occlusion probe rays provide a *1-bounce-only* illumination effect (or rather, anti-illumination effect, since it is always subtractive), Final Gathering in mental ray can provide *diffuse multibounce* illumination.

Furthermore, the Final Gathering algorithm in mental ray is very much enhanced compared to earlier versions of mental ray and resolves many issues related to potential animation flickering and detail loss, closing the gap even further between Ambient Occlusion and Final Gathering, making it more an artistic and pipeline-architectural choice which method is employed.

1.6.1 Tips

Occlusion is a raytracing process therefore it requires activation of the raytracing algorithm, **trace on**. The same of course applies to Final Gathering, as Final Gathering rays are a subset of ‘trace’ rays. Since we are walking through the kingdom of raytracing, the acceleration structure is important. Hence, take a look at the BSP size and depth parameters in your options block, check their average value from mental ray feedback messages, and eventually modify the defaults. Also note that the distance parameter in the `mib_amb_occlusion` shader is very important because it limits a probe ray’s length, allowing it to return the environment color after the specified distance. This reduces memory footprint and speeds up the algorithm flow.

The `mib_amb_occlusion` shader may appear unable to handle a closed scene. A fully enclosed scene is by definition fully occluded. So, in such situations, try limiting the distance of the probe rays. To only detect surfaces within a certain distance, you must set the `max_distance` parameter. It defaults to 0, which means “infinity”.

Note that Final Gathering has a similar control called *falloff range*. As noted previously, this can also help to reduce the memory footprint.

1.7 Ambient Occlusion using Final Gathering

In mental ray the Final Gathering algorithm can return the scalar occlusion value, therefore one can also calculate actual Ambient Occlusion by using the Final Gathering algorithm. This ignores any bounced light and simply uses the preprocessed Final Gathering points and the Final Gathering filtering to generate a grayscale occlusion result, sort of a “best of both worlds” approach.

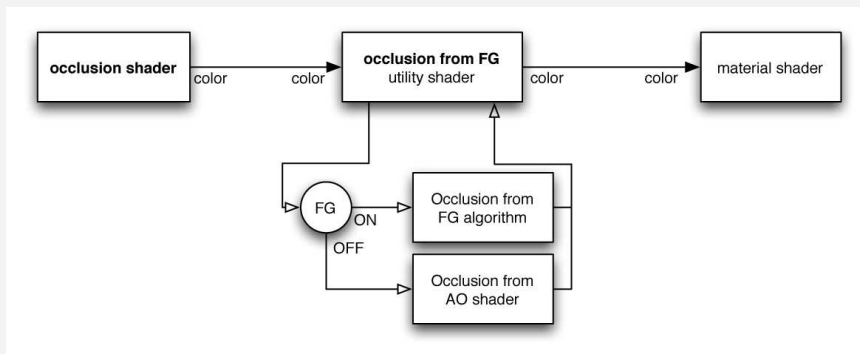
We have previously introduced that the `mib_fg-occlusion` shader can be used for this target.

Note that calculating Ambient Occlusion with help of the Final Gathering algorithm is different than using Final Gathering directly, since no actual bouncing of light occurs, only occlusion. The difference is that the precomputed Final Gathering points and the smoothing and filtering algorithms of Final Gathering are used.

The single parameter **`result_when_fg_is_off`** becomes the return value when final gathering is off. When final gathering is on, the occlusion value is returned as a grayscale value, and the parameter is never evaluated at all or used in any way.

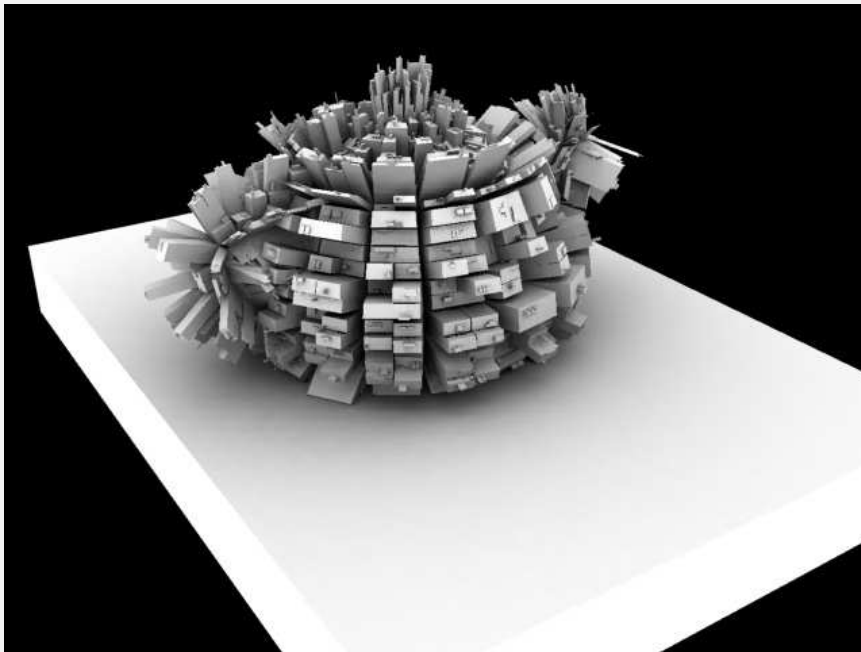
Note also that if one puts `mib_amb-occlusion` into the `result_when_fg_is_off` parameter, the result will have occlusion calculated by final gathering when it is on, and occlusion calculated by `mib_amb-occlusion` when final gathering is off. Obviously that is just one of the possible uses: you can connect *anything* to that parameter.

The following diagram shows the shading graph logic with the occlusion shader:



Occlusion from Final Gathering or from the occlusion shader

The following image is the result of calculating Ambient Occlusion with the help of the Final Gathering algorithm through the `mib_fg_occlusion` shader:



Ambient Occlusion calculated with the help of the Final Gathering algorithm through the `mib_fg_occlusion` shader

Therefore, together with the two images of the previous chapter we can now obtain three kinds of occlusion effects:

FG occlusion from plain Final Gathering (an additive effect which adds bouncing color, can be a single bounce or multibounce⁶)

AO occlusion from `mib_amb_occlusion` (subtracting occlusion from assumed omni-present environment light)

AO-FG occlusion from `mib_fg_occlusion` with FG turned ON (where the Final Gathering algorithm calculates occlusion)

©2005 mental images, all rights reserved. Tutorial by Paolo Berto, with contributions from Zap Andersson, Barton Gawboy and Matthias Senz.

⁶mental ray only