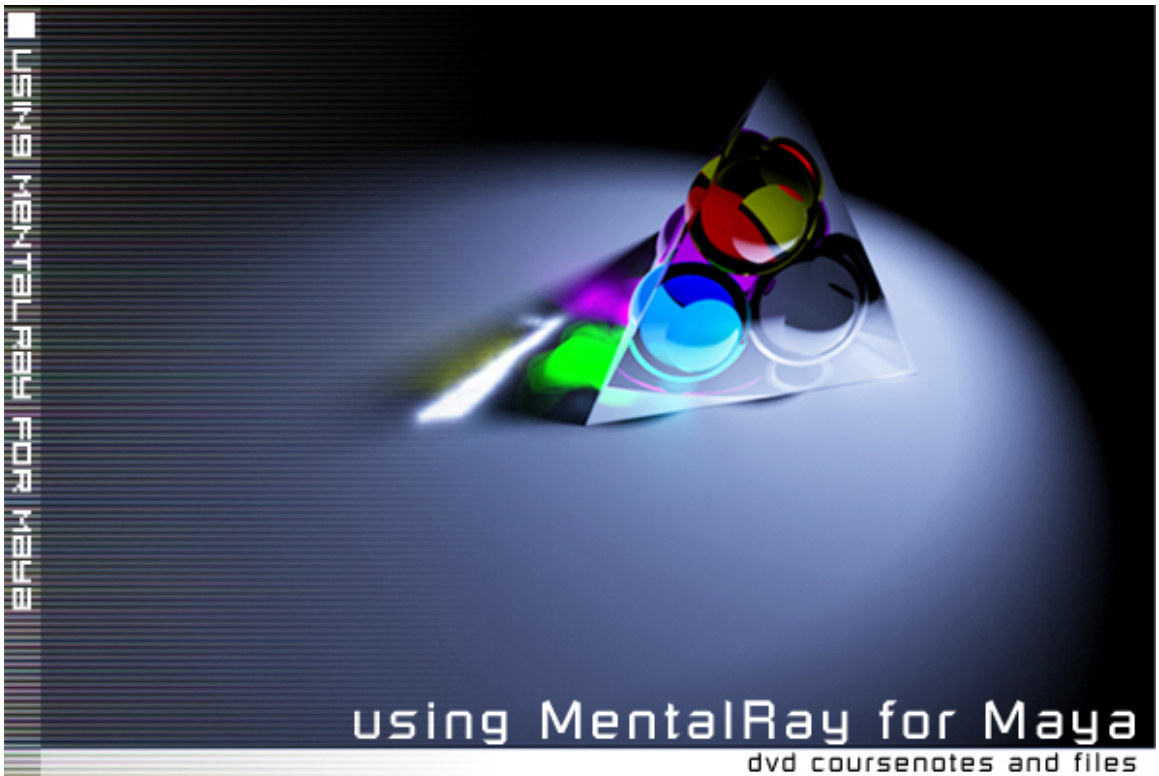


using MentalRay for Maya

dvd coursenotes and files

version 1.0

based on: MRfM 1.5 gold / Maya 4.5 gold
material by: paolo berto
mailto: pberto@jupiter-jazz.com
web: www.jupiter-jazz.com

contents:

caustics
global illumination
final gathering
real nonzero arealights
fine displacement
HDRI techniques
light baking
motionblur

EXTRA
blurred reflections
contours
faked glow
volumetric fog
volumetric caustics
antialiasing guide

using mental ray for Maya DVD course note documentation

prepared for 3December 2002 Masterclass and Using mental ray for Maya DVD

written by: Paolo Berto
mailto: pberto@jupiter-jazz.com
web: www.jupiter-jazz.com

based on:
mental ray for Maya 1.5 gold
Maya 4.5 gold

mental ray for Maya download:
www.aliaswavefront.com/Maya/getmental

FAQ:
http://www.aliaswavefront.com/Maya/support/faqs/

Index of the contents

Introduction

- presentation of the teacher
- purpose of this documentation
- level of knowledge required
- software requirements and 3rd part material used

From the Maya renderer to mental ray

- analogies
- differences
- render comparisons
- advanced functionalities
- mental ray rendering process
- Maya rendering process
- limitations and differences

Using mental ray for Maya

- installation/loading
- interface, the unified rendering workflow
- the photon map approach
- caustics
- global illumination
- final gathering
- real nonzero area lights
- fine displacement
- HDRI techniques
- light baking
- motion blur

EXTRA

- blurred reflections
- contours
- faked glow
- volumetric fog
- volumetric caustics
- antialiasing guide

Goodbye, special thanks and further reading

presentation of the teacher
Paolo Berto worked in A l i a s | wavefront for 2 years as a as a support product specialist in Belgium.
He supported and consulted many customers both in Europe and US.
Following closely the mental ray for Maya project he has provided internal support and most of the related content material used by A|W.
Paolo specializes in rendering techniques, lighting and shading network and he's comfortable with many rendering engines.

purpose of this documentation
First purpose of this documentation is to show the integration of the mental ray rendering application developed by Mental Images with A l i a s | wavefront 3D character animation software Maya.
This integration is represented by the “mental ray for Maya” plugin (from now on mrfM) in its current version 1.5 (a superset of the mental ray version 3.1 standalone renderer).
This integration has been permitted by two key factors:

- The power and flexibility of the Maya API software development kit
- strong similarities between the common DAG (directed acyclic graph) of the Maya and mental ray scene description database architecture

Second purpose is to look at the special features that the use of mental ray offers in extension to the capabilities of the Maya renderer.

level of knowledge required
The level of knowledge required to follow this material is “beginner to intermediate”, you do need to know something about Maya and specific further knowledge about the Maya rendering workflows and terminology.
I will include some extra terminology to understand properly the mental ray architecture when necessary. I strongly recommend reading the mental ray reference manual included with the online documentation.

software requirements/3rd part material
You need Maya 4.5 gold and mrfM 1.5 publicbeta or gold. Some of the HDR textures I’ve used are freely available at www.debevec.org where you can also find HDRShop in order to manipulate/transform them.
I used also a 3D plane F-5E Tiger II, it is freely available at www.meshfactory.com while the model of the cartoon shader chapter is courtesy from mr Bruno Pollet www.br1.org which also is the author of the final gathering animation showed in the DVD and included in the DVD.
Finally the non-periodic caustics textures are from Jos Stam at:
http://www.dgp.toronto.edu/people/stam/reality/Research/PeriodicCaustics/index.html

FROM THE MAYA RENDERER TO MENTALRAY

analogies
the Maya renderer *is* Maya:
the integration between the Maya renderer and Maya is not simply apparent, this integration is present both in the UI and at the (open) architecture level. This provides flexibility both at the programming side, via the API, and at the user side, with a series of expression-like utility nodes and an integrated MEL scripting language. Such a building block system allows a complete visual interface to shaders creation. And this interface is completely supported by mental ray for Maya.

To describe shading, the Maya renderer uses dependency graph mechanism called the “DG” (complete with API for user-written nodes). A shader’s look is defined by a DG network and the shading process is the evaluation of this network.
There is one extension to the basic DG to allow contextual information to flow between different nodes: this is a special network evaluator based on “implicit connections” which are present only when a “real” connection doesn’t exist.
So shading has its own evaluator to traverse the network. The Maya DG is interpretive and it doesn’t take advantage of invariance in the network. It’s a 2 pass process that first partially evaluates the network factoring frame constants and then finishes in a faster way the evaluation on a shading sample base.

There are fundamental similarities between Maya and mental ray scenes. They both define a hierarchy explicitly by the use of a directed acyclic graph (DAG). Consequently, it is a simple matter to traverse the Maya DAG, creating mental images (mi) entities corresponding to the type of each Maya shape node visited. This will recreate all geometric objects, lights and cameras found in the scene. Everything mental ray needs to start rendering the scene.
We’ll see later what happens when mental ray starts rendering a scene.

On a more general view there are also architecture analogies:

- they’re both hybrid multithreaded raytracers (the 1st “ray” is generated via the scanline algorithm - the word “ray” here is a generalization, because scanline deals with 2D space - then if necessary raytracing is called for successive ray generations. Note that in mental ray if there are lens shader calls, scanline shuts off automatically)
- they both use ray marching for rendering volumes
- as previously introduced they both have a DAG hierarchy (DG is the dependency graph, a more general graph to which the DAG is part of), a hierarchy of nodes connected between input and output connections. Every element in Maya is described by either a single node or a series of connected nodes, each node is described by an external interface (a series of attributes related to what a node is designed to accomplish). These nodes can be inter-connected with connection between attributes. These connections are known as dependencies.
- they both build an acceleration structure for secondary rays (Maya performs this in the “rendering a tile, shade geometry phase”) which, also supports scanline rendering.
- they are both implemented in an independent way and can work on diversified machine architectures.
- they both support an optimization to memory map textures: BOT textures in Maya and .map textures in mental ray.

differences

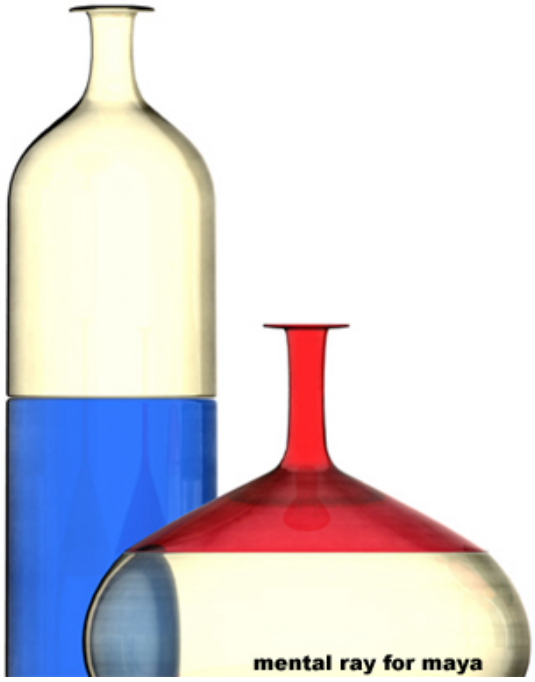
- mental ray exists also as a standalone renderer while the Maya renderer is part of Maya hence its memory footprint is bounded below Maya's. The mental ray standalone is separate. mrfM being an integrated plugin renderer acts exactly as the Maya renderer. The batch renderer, like the Maya one runs as a separate process
- Maya provides for NURBS explicit initial and secondary tessellation attributes, a base triangulation is first performed and then further tessellated until the secondary set of conditions are met. In mental ray there's "only" one approximation statement for a surface, initial tessellation can be compared to "parametric" approximation, secondary tessellation to LDA method (length/distance/angle). When the "derive from Maya" flag for tessellation is used, tessellation is automatically calculated trying to match the same level of quality by mrfM.
- Maya completely separates visibility, shading and temporal computations. Introducing a specific antialiasing method for each of them. Two specific techniques called EAS (Exact Area Sampling) to resolve visibility and "atomic super sampling" a form of more flexible adaptive super sampling which takes into account the fact high frequencies information in different domains could not be related: it solves each problem domain separately to avoid extra computations (for example geometric spatial and temporal aliasing are split, and the user can decide how to resolve a quality issue changing all of them on a object basis). So Maya's primary visibility computation is based on higher resolution screen masks, the point sampling nature of raytracing implies that primary and secondary geometric antialiasing quality may be inconsistent. Maya relies on adaptive antialiasing in secondary illumination effects (on an object basis), while mental ray uses a low sample approach. In Mentalray jittering, motion blurring, area light sources, and the depth-of-field lens shader are based on multiple sampling that is based on varying the sample locations in time, 2D or 3D space. mental ray offers a proprietary implementation of the Quasi-Monte Carlo method for achieving these variations. Sample locations in time, 2D and 3D space are deterministically chosen on fixed points that ensure optimal coverage of the sample space. The algorithm is similar to fixed-raster algorithms, but avoids the regular lattice appearance of such algorithms. The resulting images are identical if the scene is re-rendered with the same options due to the deterministic nature of the algorithm. Quasi-Monte Carlo methods can be succinctly described as strictly deterministic sampling methods. Determinism enters in two ways, namely, by working with deterministic points rather than random samples and by the availability of deterministic error bounds.

renderers comparison

Since its first beta version, mental ray for Maya has been able to "interpret" many of the Maya rendering nodes, and with version 1.5 more features are supported and basically all rendering nodes are

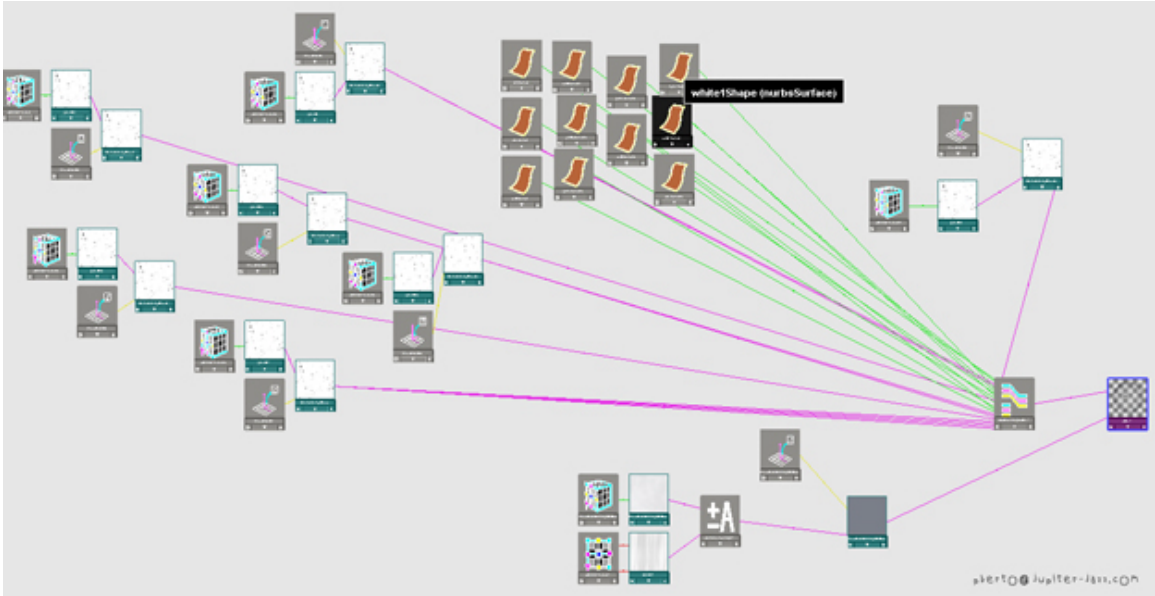
working correctly.
A good example is this raytracing scene with light coming from a single wide area light casting raytracing shadows. These hand blown vases called “Bolle” were originally designed by a Finnish designer called Tapio Wirkkala in 1968 for the famous factory Venini. With mrfM v1.0 beta the scene was rendering fast and without any problem.

see also the two renderers side by side:



and in detail:

see also the shading network: this scene has only 1 blinn shader for all the surfaces, then different attributes are switched over different sub-networks, basically these sub networks are driven by samplerInfo in order to define for each bottle different glass properties.



advanced functionalities
the use of mental ray for Maya allows for further advanced functionality to the feature set available with the Maya renderer:

- using the photon map approach global illumination and caustics are supported.
- Final gathering.
- real nonzero area lights.
- subpixel displacement.
- unclamped value texture (HDR, floating point TIFFs) input and output.
- motion blur of everything.
- incremental changes for animations.
- host parallelism.
- participating media for volumetric effects.
- Phenomena.
- cartoon rendering.
- and other special mental ray standalone features via MR nodes.

mental ray rendering process

So we have seen that the architecture similarities and Maya powerful API SDK allow for easy Maya scene recognition into mental ray., But what happens when mental ray starts rendering the scene?
Well the nice thing is that we can always see what the rendering is doing, even on the tiniest detail. This is possible via the option “export verbosity”:

render globals> translation> export verbosity

from no messages to detailed messages, in order of how much information is passed on the output window. All this information can be very useful to debug your scene, take in mind that with detailed messages there’s a slowdown in render performance.

A typical mental ray rendering process could be summarized in this way:

1. via the Maya API: construction of the scene graph (before the Maya scene can be rendered in mental ray, it must be processed into a “mi stream”.
2. traversing the scene graph for construction of geometry and eventual generation via geometry shaders. Then marking of the objects for tessellation (traverse the Maya DAG creating mental images (mi) entities corresponding to the type of each Maya shape node visited).
3. tessellation of objects which have not been tessellated or have changed since last tessellation.
4. shadow map computation (if required or not already present on disk).
5. photon map computation (if caustics or GI are required and maps are not already present from disk).
6. final gathering pass.
7. acceleration of data structures for secondary rays (eg: BSP/GRID) if raytracing is activated.
8. rendering image on a sample basis and depending on settings: scanline initially and for subsequent transparency then raytracing for reflections and/or refraction rays using previously generated shadow/photon maps or disk stored maps.

9. running output shaders if specified and final writing of output image .

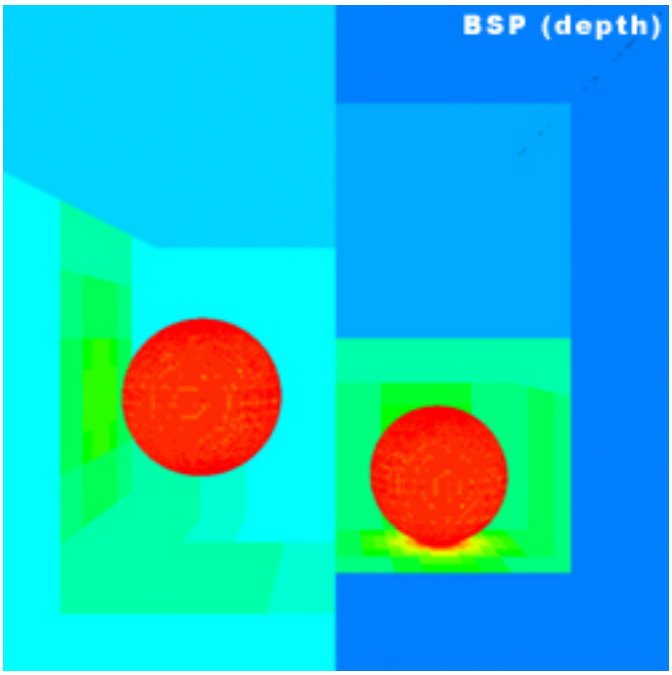
Note: these steps may overlap in some cases, incremental changes are exploited plus the renderer uses cached info from previous renders if possible. If lens shaders are used raytracing will be used for all rays and the scaline algorithm will be shut off.

Diagnostic tools to evaluate your renderings, can be found in:

render globals> quality settings> diagnostic

you can find useful tools to evaluate various aspect of the rendering process (distribution of irradiance, density…) and also evaluation of the acceleration structure you're using: BSP (binary space partition) or GRID (renewed algorithm starting from mental ray v3.0).

The following image (a GI Cornell Box) shows the diagnostics of the BSP depth, BSP is a 3d space subdivision in a nested set of voxels, a sort of tree where each leaf contains triangles, hence leaf size and depth have great importance for its memory efficiency, memory limit can be set though. In this tree-structure you refer to leaf size (the max number of triangles per leaf) and tree depth (the max nested level). Max depth 25 is good for small scenes, 40 for medium and 50 for large ones.



As for the GRID acceleration: it equally subdivides 3d space in isovolumetric voxels.
Mentalray 3.x uses a hierarchical grid algorithm which has a depth and grid parameter to further subdivide voxels if too many triangles are in them. It has a comparable speed BSP and a better memory usage, but it's scene dependent (you should use it in scenes with heavy geometry

uniformly distributed). Small values give rendering speed but high memory usage while large values give preprocess speed with less memory usage.
Following the same image as before rendered using the GRID raytracing acceleration and it’s relative diagnostic mode in camera space coordinates.

Maya rendering process

I include also the Maya rendering process, you can find more details on the architecture of the Maya rendering in a paper called “Design and Implementation of the Maya Renderer” written by the original Maya rendering team: Kelvin Sung, James Craighead, Changyaw Wang, Sanjay Bakshi, Andrew Pearce, and Andrew Woo.

1. Read geometry from Maya scene database. Only geometry in the viewing frustum is considered, object which participate in raytracing are always considered
2. Computation of depthmap shadows. For lights which intersect with viewing frustum with compression of the map and decompression on demand.
3. Building of the image tiling system, to maintain a constant memory footprint, using a quadtree-subdivision algorithm threshold controlled
4. Scan conversion for a tile: until this point the only computations were related to objects bounding boxes. In this phase the renderer with a process from front to back, and only for visible objects and one-time per multi instances performs: initialize object shading state, tessellation (lazy), scan convert triangles into the screen mask

5. Render a tile:

   a) computing visibility, if motion blur present pointsampling triough time
   b) shade geometry, here idf secondary rays are necessary a bsp accellleration structure is built lazily
   c) computation of volumetric primitives, which do not participate to the screen mask visibility operation. Instead placeholders of volumetric primitives are inserted into data structures. Particle systems are considered a subclass of the volumetric primitive type.
   d) Adaptive extra shading samples. Extra shading samples are taken on a per object contras. No adaptive super sampling in time.
   e) multi pixel reconstruction via the multi pixel filtering

Limitations

Here I will include a short description of what is not possible to render with mrfM 1.5 within the Maya UI, workarounds are made possible by the use of mental ray standalone and in some cases special libraries. I remind you that you may find a more updated and exhaustive list on the online manual and at the public FAQ at:
http://www.aliaswavefront.com/Maya/support/faqs/

- all Maya post-processing effects: Paint Effects, shader/light glow, optical effects, 2d motion blur, and fur rendering. There is a simple workaround using contour shaders for a “faked glow” effect (see the extra section).
- software rendering of particle systems (particle instancing is instead supported)
- direct rendering of Maya subdivision surfaces. Subdivision surfaces are supported by mental ray standalone when the library “libmisubdiv” is linked in. This library implements the ‘Loop’ subdivision scheme for triangle meshes, and the ‘Catmull-Clark’ subdivision scheme for quadrilateral meshes. C/C++ API for subdivision surface construction and manipulation are also available, this API can be used by geometry shaders to create renderable subdivision surface objects. Check the mental ray reference manual for more details
- interactive photorealistic rendering (IPR). As long as it’s not supported you could IPR tuning in Maya and then render in mental ray, but only for Maya and mental ray common features
- Field rendering
- Pre/Post Render Mel scripts
- Fluid effects
- Custom mental ray text nodes cannot be interpret in the integrated renderer, they can however be exported to .mi format and rendered with the standalone renderer

other limitations/differences:

- no stereographic rendering,
- DOF differences
- Limitations with layered shaders and bump x layers and layered shader with GI
- File texture has no prefiltering and supports only mipmap filtering,
- Bump 2d and 3d have view dependent filtering
- Place3d texture uses only the transformation matrix
- Render stats> receive shadows flag is not recognized

USING MENTAL RAY FOR MAYA

Installation / Loading
After having installed the plugin we can load it via:

window> settings/preferences> plug-in manager> Mayatomr

clicking on the (i) icon will show the current mrfM plugin version and the Mentalray version in use: 3.1.2.13 in the publicbeta case and 3.1.4.1 in the final (gold) release.
You can also see the list of new nodes we can use/create.
See the “extra” section for activating UI visibility for custom mental ray nodes at startup.
One last thing: to set mental ray as the default renderer in Maya, go to:

window> settings/preferences> rendering

Interface, the unified rendering workflow

What has been done here is called “unified rendering workflow”, basically once chosen
“mental ray” in:

render> render using> mental ray, or in the renderview cascading menu.

You’ll find that all the attributes you were used to tweak when rendering with Maya are just interpreted and rendered in the Maya renderview by mental ray for Maya in a total transparent way, plus the new attributes for advanced features are placed coherently in the relative attribute editor (AE) and eventually in new sections of it.

One of the most important things is that by default mental ray will derive rendering settings from Maya without any user’s action, you will see many “derive from Maya” flags in various AEs and in render globals while getting confident with mental ray for Maya: they are normally switched to “on” by default, for example tessellation of NURBS, which in Maya is a 2 phase approach, is automatically derived to the

way mental ray tessellates geometry providing equivalent quality. Same discussion for displacement, which, even if approached in many different ways in mental ray, outputs an equivalent of the feature-based displacement mapping of Maya.
Switching off “derive from Maya” flags, allows you to control manually each particular attribute of the mental ray interpretation of the Maya scene database. This is very useful for more advanced users requiring custom flexibility, in this case the mental ray technical reference manual becomes very handy.

To conclude: basically all Maya rendering and utility nodes are correctly interpreted by mental ray for Maya and you can find the new attributes in:

- various Maya attribute editors (AE)
- mental ray render globals
- Maya renderview
- file> export menu
- window> rendering editor> mental ray
- hypershade: edit> convert to file textures (mental ray)

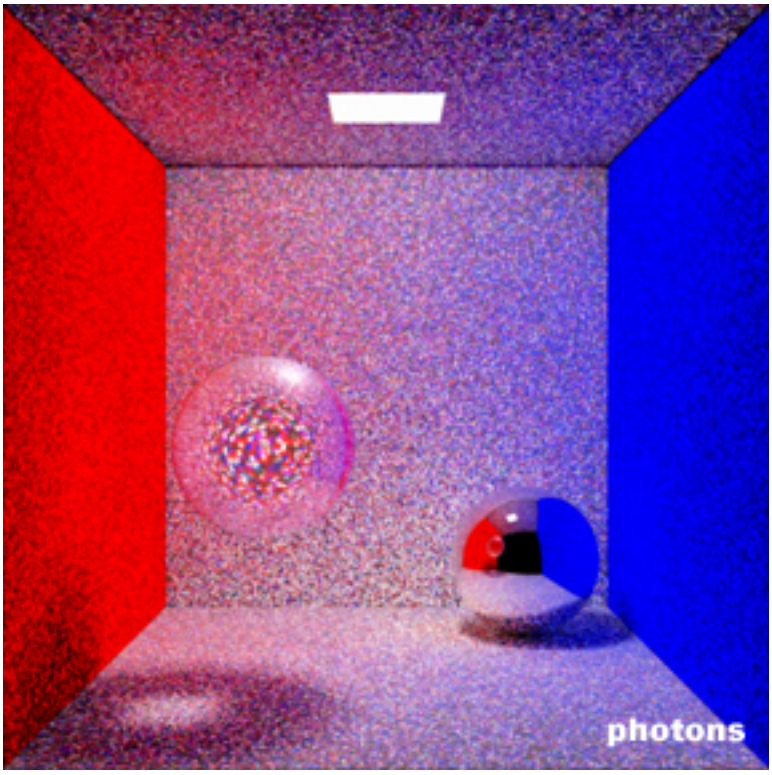
Finally our target is to render, then it’s useful to know in how many ways we can perform the action:

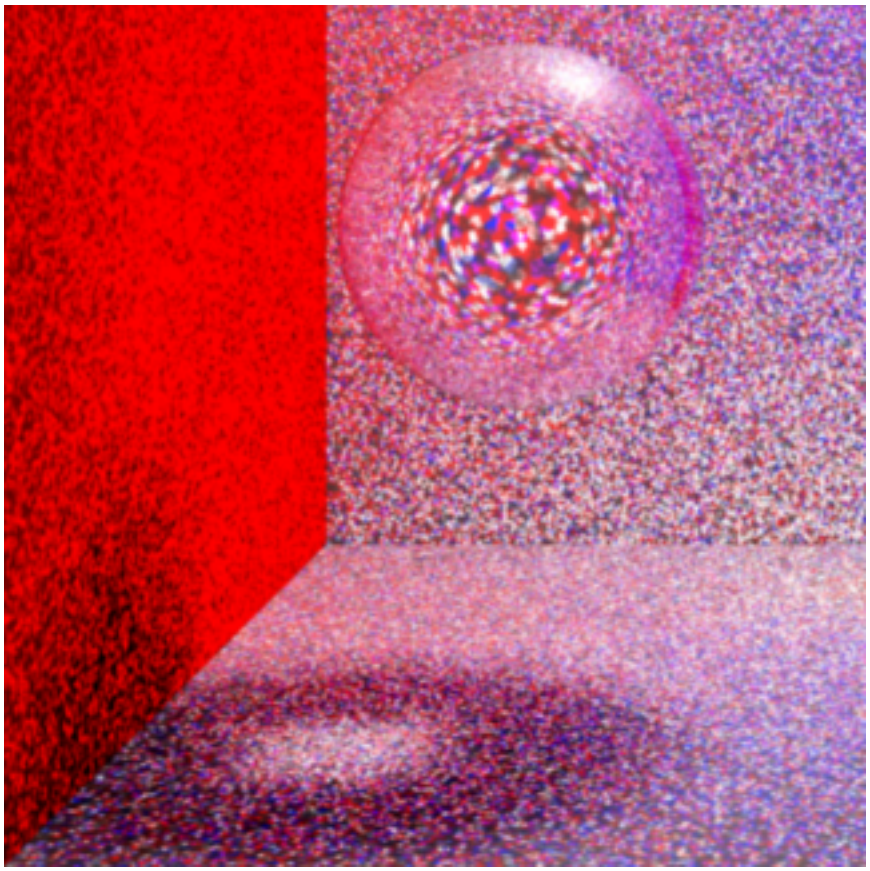
- interactively from the Maya UI
- batchrender from the Maya UI (this is a separate memory process which allows you to keep working in the Maya UI and even to render interactively at the same time, in this case a good amount of RAM and a dual-cpu machine are recommended and useful (for example you could specify to mental ray to render on one cpu only)
- batchrender via the command line using the “Mayarender_with_mr” command (see online docs for flags)
- export to the .mi format and render with the standalone renderer (if you have it)
- render lightmap textures via mental ray for Maya baking options

the photon map approach
mental ray for Maya supports caustics and global illumination simulation using the “photon map” method. A photon map is a 3D data structure holding photons, which represents a quantized flux of light emitted from an energetic lightsource(s) and then successfully stored on scene geometry. Caustics deal only with the specular component of a lightsource, and they are a subset of global illumination, which takes into account also the diffusivity.
The photon map is generated at the end of a preprocessing step in which photons are emitted from the light sources and traced through the scene using photon tracing. You can follow the steps of a photon map generation reading the rendering verbosity on the output window via:

MentalrayGlobals> Translation> Export verbosity: progress messages

The following image features a Cornell box with GI and caustics where an area light emits 1 million GI photons and 100 thousand caustic photons. Using very low accuracy and radius values results in showing little dots as photons stored on surfaces.

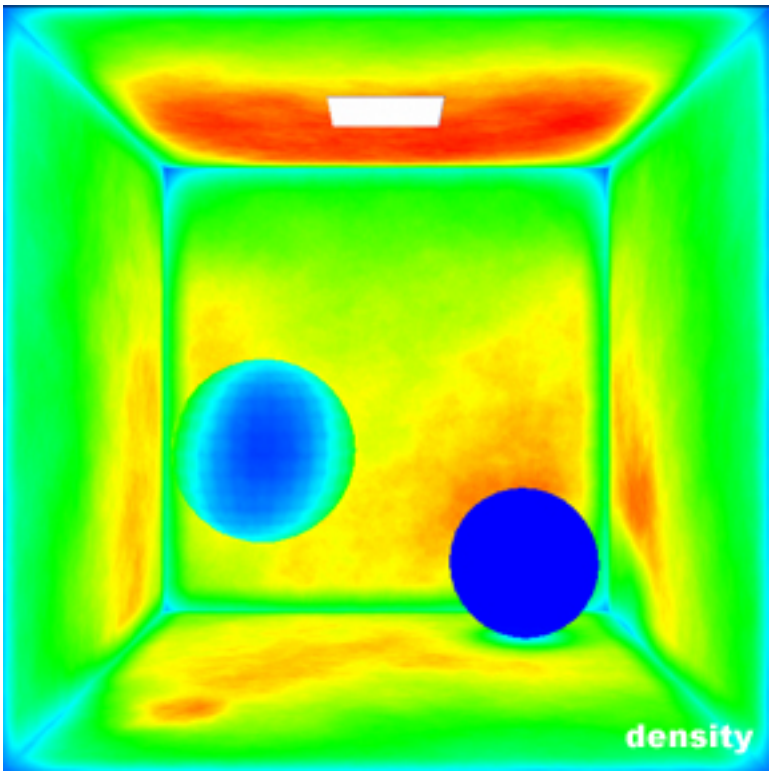


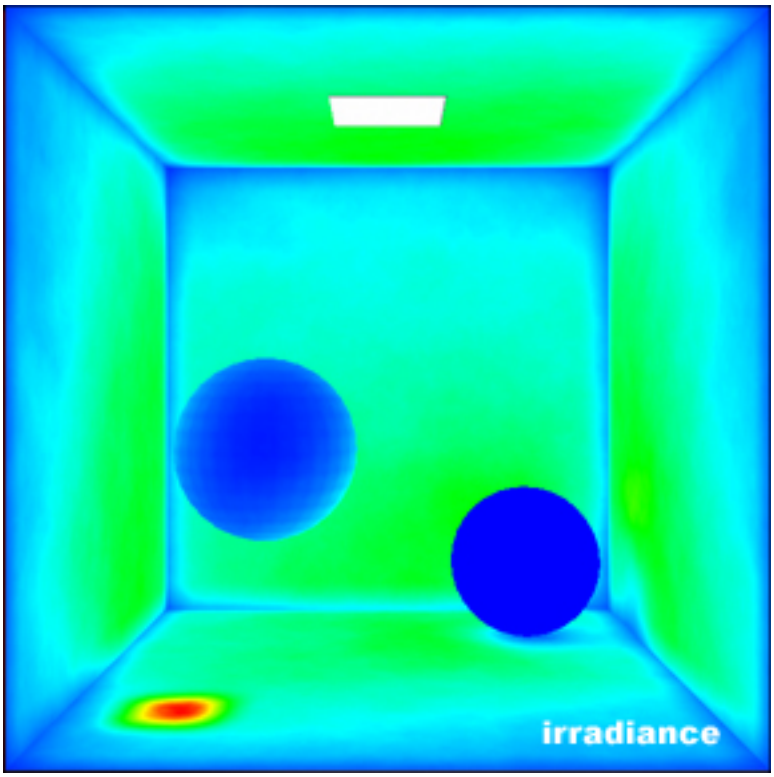


There's also an animation of the number of photons emitted by the light increasing exponentially in time. Check the "animation" directory in the related coursenotes files.
The diagnostic modes are also very useful to debug your scene, they are accessible on:

MentalrayGlobals> your-render-quality> Diagnostics

The following images shows the Cornell box in its density (number of photons x unit area) and irradiance diagnostic modes.


density


irradiance

Photon maps can be saved and reused for speeding up rendertimes for example while tweaking other non photon map dependent attributes and while rendering animations where the general illumination of the scene doesn’t change much in time.
You can find the relative attributes in:

MentalrayGlobals> caustics/GI> photonmap file
MentalrayGlobals> caustics/GI> photonmap rebuild

First, specify a filename keeping the rebuild flag on, then once the map is generated, at the next render leave the filename specified and remove the rebuild flag.

A final note about the accuracy and radius parameters:

- Accuracy is how caustics/GI is estimated from the photon map during the rendering phase. It is the maximum number of photons to consider for illumination within the specified radius.

- Radius is the radius of the circle centered at an intersection point where mental ray will search for photons.

Accuracy and radius use the information stored on the photon map at render time, hence, when you feel you have the right number of photons, you can save and reuse the map and tune them.
It is very important to optimize render times when using raytracing reflections, refractions, caustics, GI, FG. Many parameters should be tweaked in order not to waste computational time for features which won't be visible in the rendered frame, some of them are:

- the number of rays used,
- the ray depth,
- the number of photons used,
- the photon depth.

From the next chapter we’re going to look at the specific examples on the DVD. Hence the explanation will be focused only on the key points for each feature, and on special considerations. Please refer to the video for the workflows.

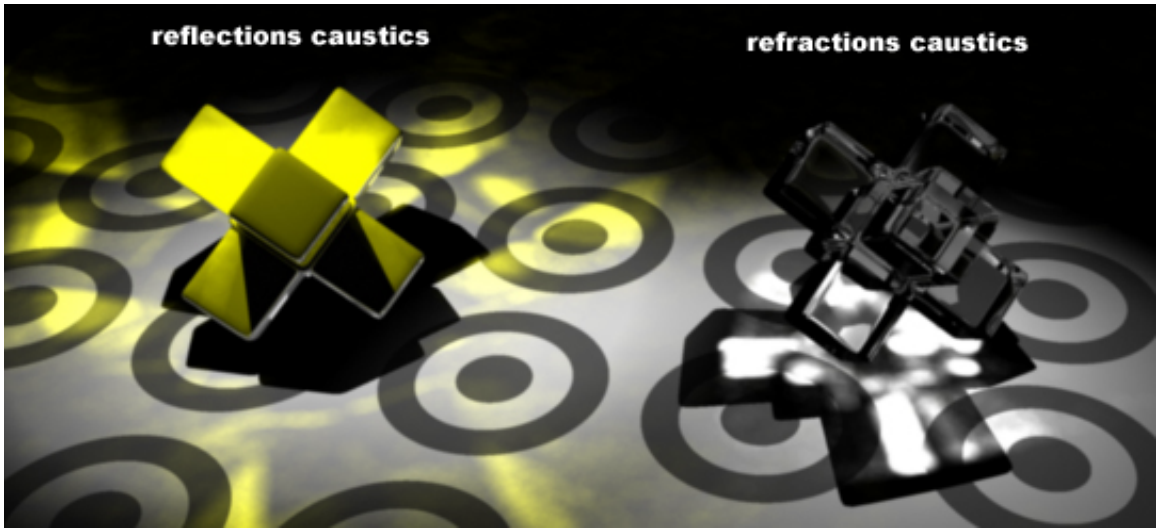
caustics

The Maya renderer does not support caustics natively, to achieve this effect until Maya 4.0, you needed the A|W Maya caustic plugin by Tom Kluyskens.

mental ray for Maya supports caustics natively. It uses the photon map method to simulate them.
Caustics are light patterns caused by specular reflections and refractions (transmissions) of light visible on diffusive surfaces, and they are a subset of global illumination.

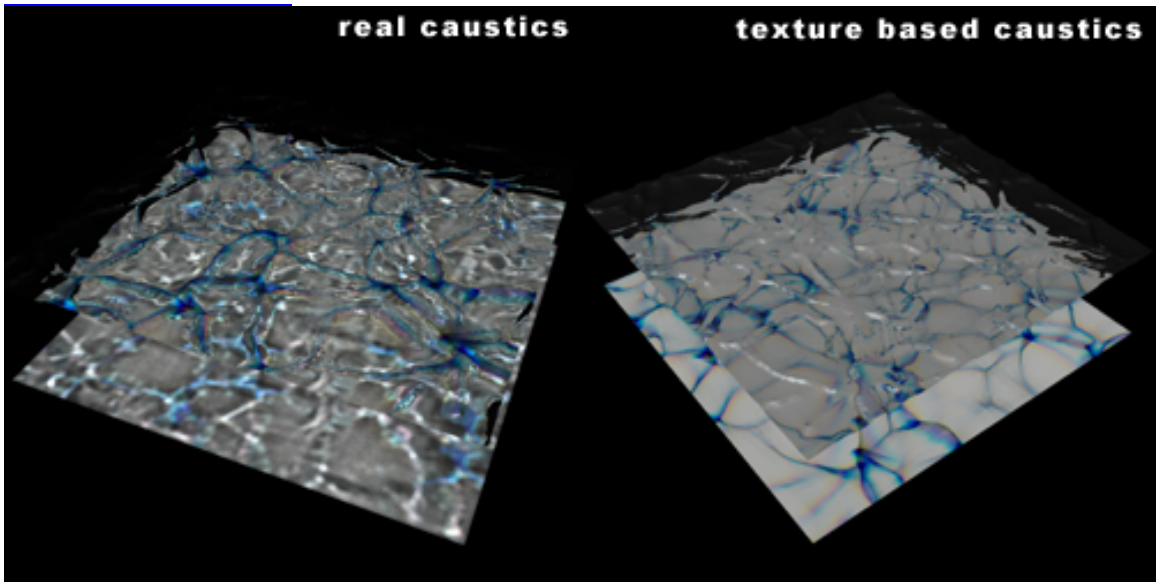
In the DVD we are looking at 2 scenes:

- caustics from reflections/refractions



- photon map caustics vs texture based caustics using the non periodic caustic textures from Jos Stam.

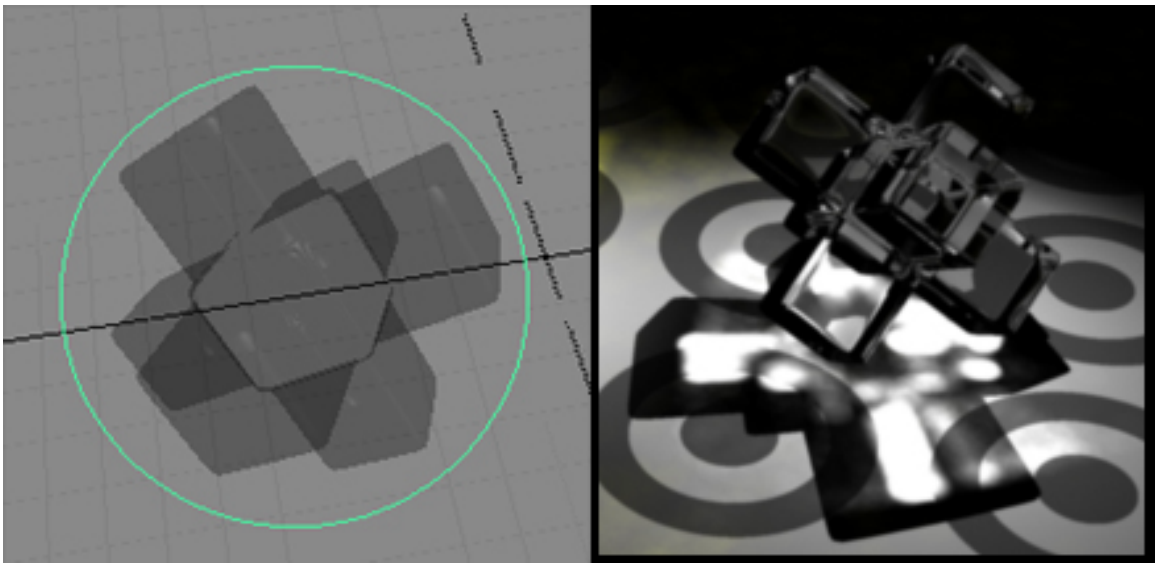
  http://www.dgp.toronto.edu/people/stam/reality/Research/PeriodicCaustics/index.html



  In the texture-based caustic scene we are using a file texture in order to represent caustics on a ground plane while the same texture is displacing the water plane (using fine displacement, see related chapter) previewed in the OpenGL view by the new utility called “height field”

some tips for caustics useful in the reflect/refract scene:

- focusing/directing caustic photon emitter lights on objects required to cast caustics. Particularly with spotlights, focusing the light on an object produces better hit-rates (stored vs emitted photons) and allows the use of less photons in order to have a much faster photon emission phase, like in the following image:



- prefer spotlights to pointlights (if your caustic casting objects are not all around your light origin). Pointlights emit photons spherically, spotlight conically hence you do require to emit more photons to get the same density (photon per unit area) on the storing surface.

- checking eventual occlusions around the photon emitter directions which could make photon bounce away before reaching the caustic caster object

- use verbosity option to check hit-rate statistics and eventual warning messages like “no photons stored after emitting n photons” on the output window.

  This is a rendering process reported on the output window when “progress messages” verbosity options are used. You can see per each light the photons emitted and stored and how many of them from specular reflection or refraction:

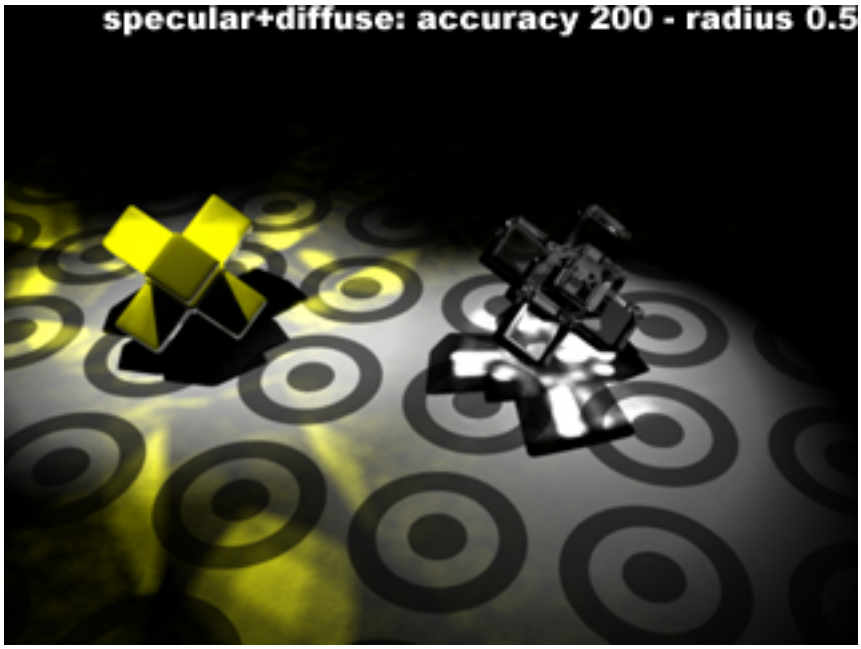
  ```
  ---
  RCGI 0.4 warn: 362004: no photons stored after emitting
  10000 photons
  ---
  RCGI 0.2 info: light #0, caustic emission,  emitted:
  549666,  stored: 0
  RCGI 0.2 info: caustic light #0: light photons 549666
  ---
  RCGI 0.2 info: light #1, caustic emission, emitted:
  24593, stored: 3642
  RCGI 0.2 info: caustic light #1: light photons 24593
  RCGI 0.2 info: caustic light #1: spec. refl. Photons 5364
  RCGI 0.2 info: caustic light #1: spec. refr. photons 16382
  RCGI 0.2 info: optimizing access to caustic Photon Map with
  3642 photons
  ---
  ```

- keeping separate the specular and diffusive light using 2 different lights to focus the specular light in order to speedup the caustic map generation and to control diffusion and eventually shadows on the diffusive light. This result in much more flexibility in controlling what you want to achieve.

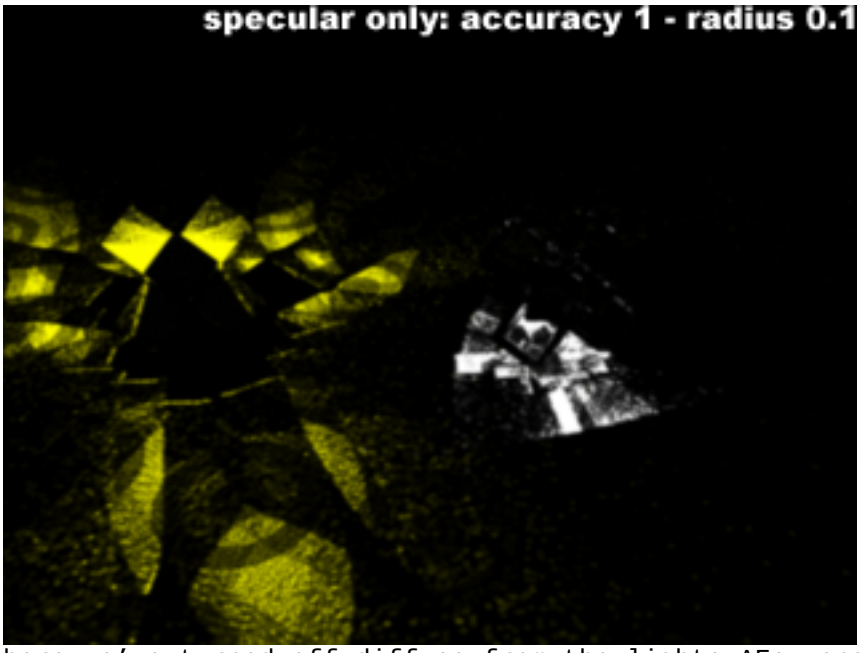
- start rendering with low accuracy and radius (see photon map approach) to see where effectively photons are stored and then increase gradually.

Now I think it’s useful to show a series of images with different values of accuracy and radius in order to understand what they effectively do.
The radius value’s main function is to define where to search for photons, accuracy defines how many photons to consider within the area defined by radius at any sample location.
They must be used together to reduce spottiness (typical with low radius settings) without blurring too much and losing global illumination (high radius high accuracy).

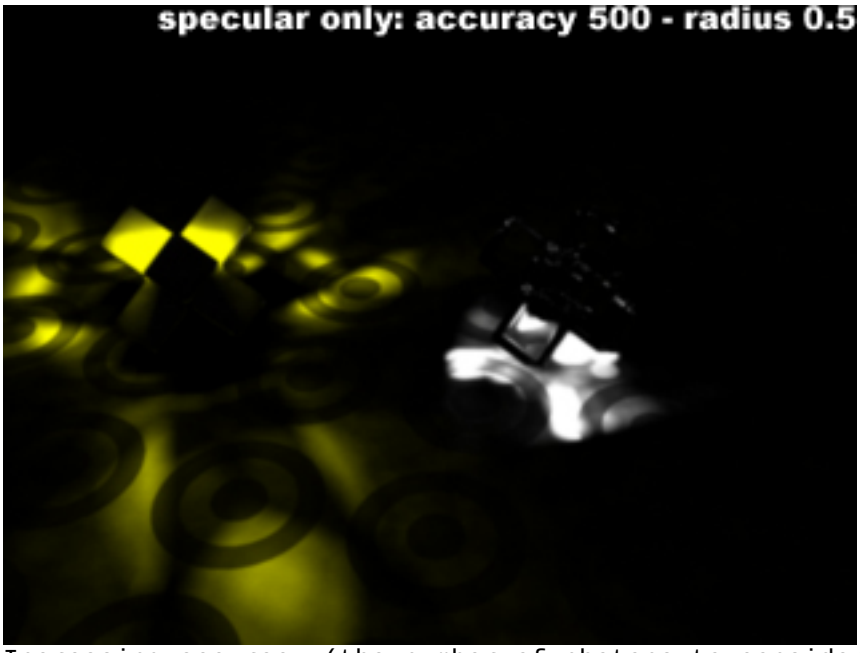
The target is to reach equilibrium between these two extremes avoiding spottiness and blurring to reach a physically correct (if required) simulation. See also the global illumination section with another serie of images.



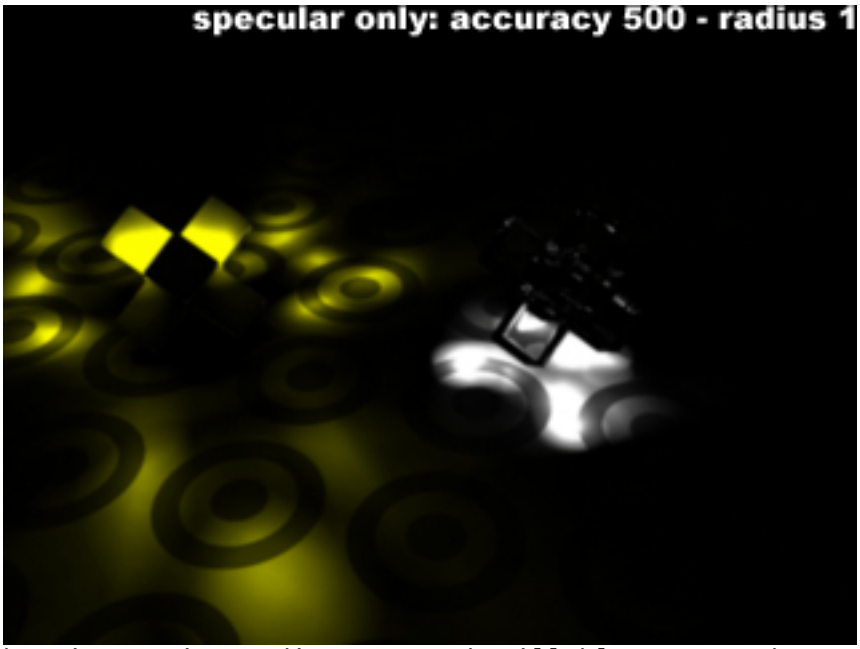
This first image features both specular and diffusive lighting (as you can see in the video, illumination is splitted) with accuracy 200 and radius 0.5, in this case it was a good compromise between spottiness and blurriness with a very low number of photons and a fast rendering.



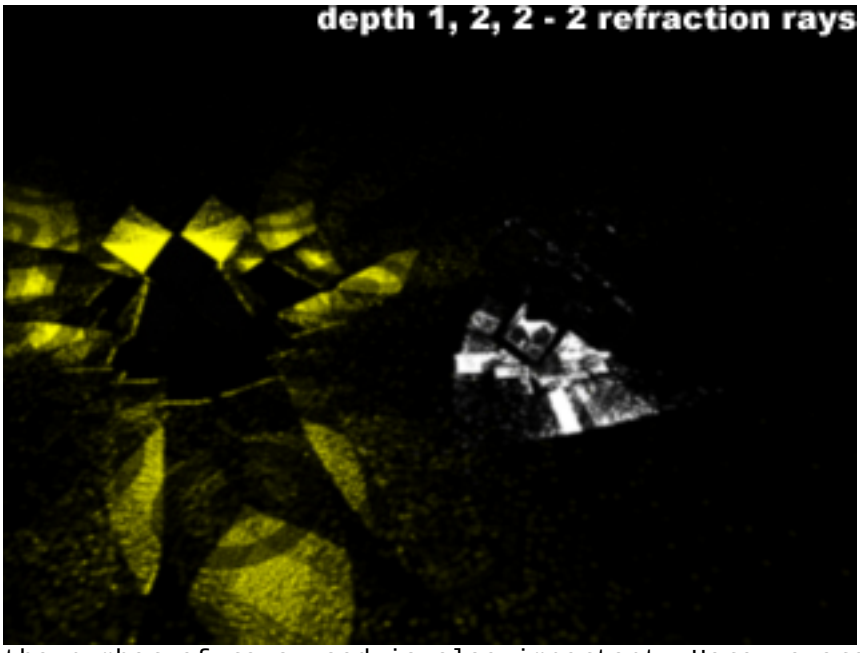
here we've turned off diffuse from the lights AEs, and we basically see the photons stored. With accuracy 1 and radius 0.1 we can see the photons themselves stored on the surfaces but the result is too dotty.



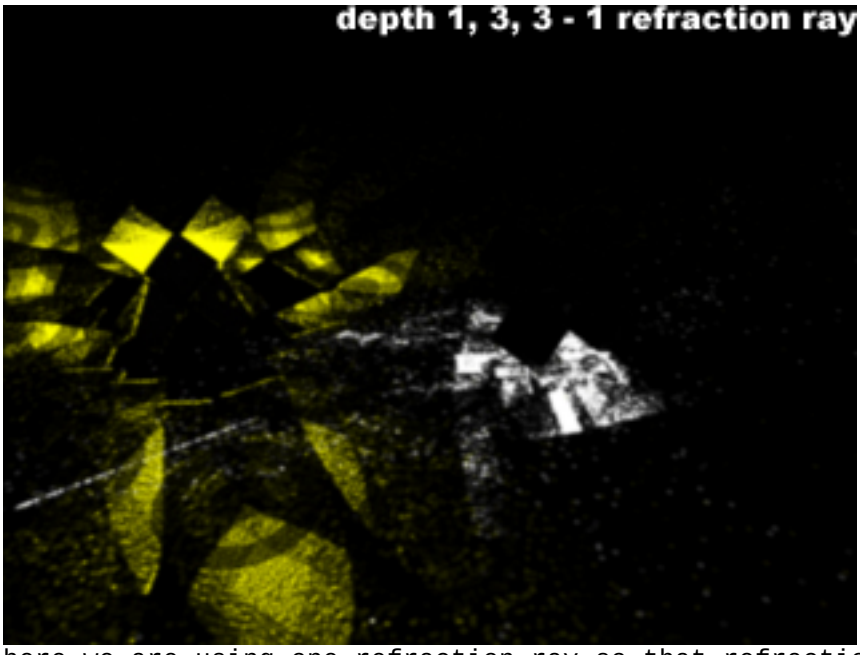
Increasing accuracy (the number of photons to consider into the radius) and the radius will eliminate the little dots because we are computing illumination on a broader area.



but increasing radius too much will blur too much as well



the number of rays used is also important. Here we are using 2 refraction rays in fact, we can (hardly) see a back-refraction on the top of the glass object. (see some small white dots over the glass object)



here we are using one refraction ray so that refraction disappears but we are using more reflection rays, notice the white caustic dotted stripe on the bottom left of the image. The following parameters are useful to control reflections, refractions and caustics patterns:

- trace depth: how often a ray can be reflected, refracted and the sum of the two
- photon trace depth: same thing but for photons
- accuracy: max number of photons considered for computing irradiance
- radius: max distance where to search photons for computing irradiance

3.4 - global illumination (and final gathering)

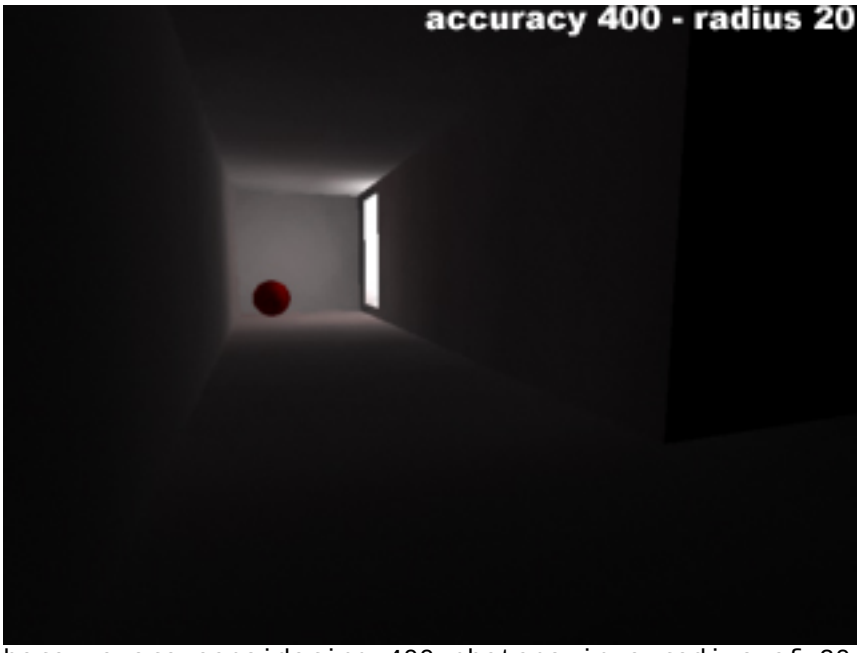
The Maya renderer supports direct illumination, meaning that the light reaching a given point on a surface has been following a direct and straight path from a light or from a transparent shadow-casting object without reflections or refractions which invokes raytracing. This is basically what is called a hybrid scanline/raytracing renderer.

mental ray works in the same way but via the photon map approach supports both global (inirect) illumination and caustics independently. Global illumination simulates *all* indirect light paths in a scene in addition to specular ones (caustics, which are then a subset of GI), and a final gathering step can be used to add a one-bounce-only indirect illumination pass which contributes to an highest physical coherency and reveals fine details (at the price of more computational time).

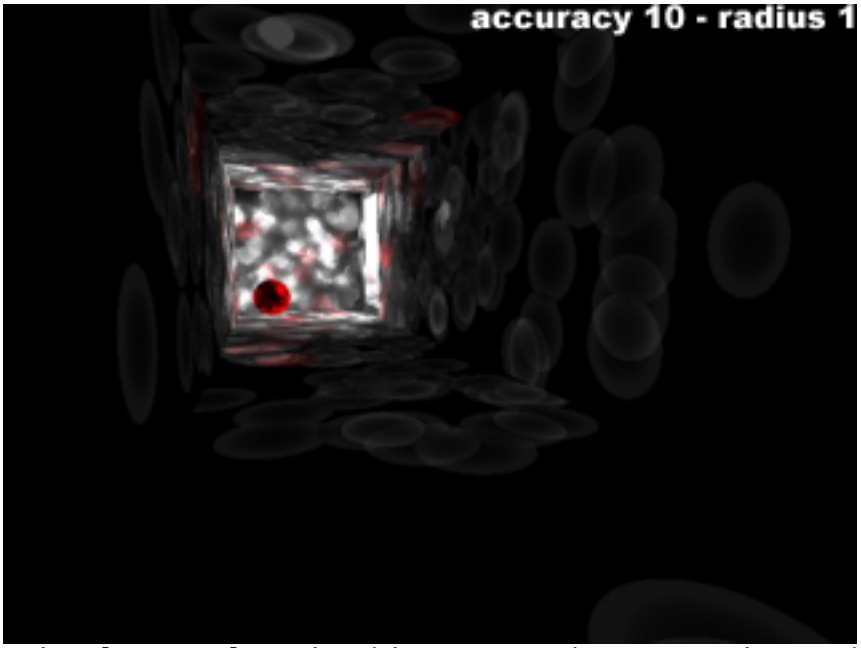
Global illumination can be used in all the situations where indirect light (both diffuse and specular) and color bleeding (an effect of GI which causes surfaces to receive the tint of their surroundings in a coherent way) are required.

In the DVD we are looking at two scenes:

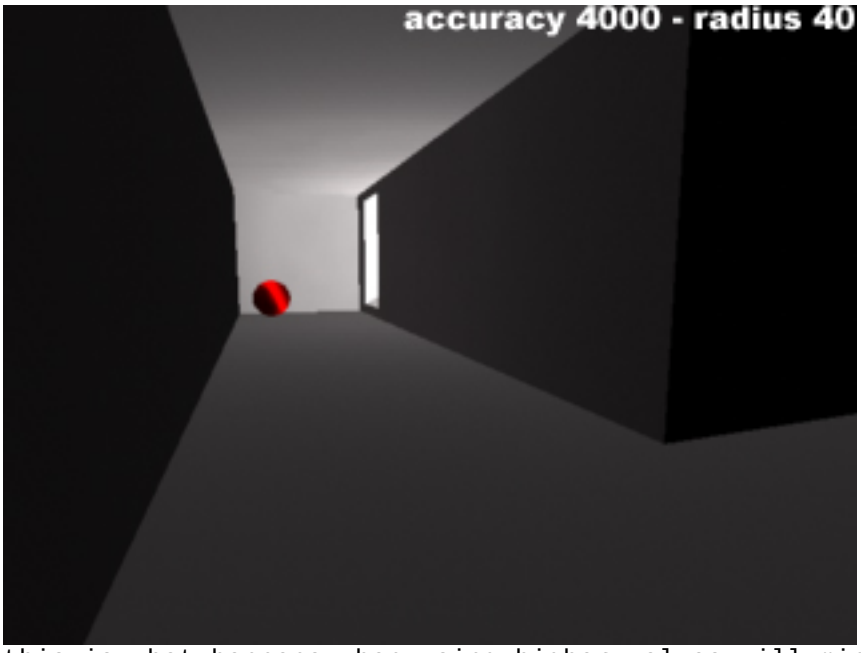
- a basement scene with a distant arealight coming in from a door which is useful to show both indirect illumination and color bleeding.
  The following images shows how the change of accuracy and radius parameters for GI are affecting the rendering:



here we are considering 400 photons in a radius of 20 units (coherently with the dimension of this basement) for indirect illumination. The scene is extremely simple, and shows how the illumination decreases with distance from the door by the absorption of photons due to the diffusive walls. The reddish color under the sphere represent the color bleeding. I want to remind you I’m using a low number of photons in the scene.



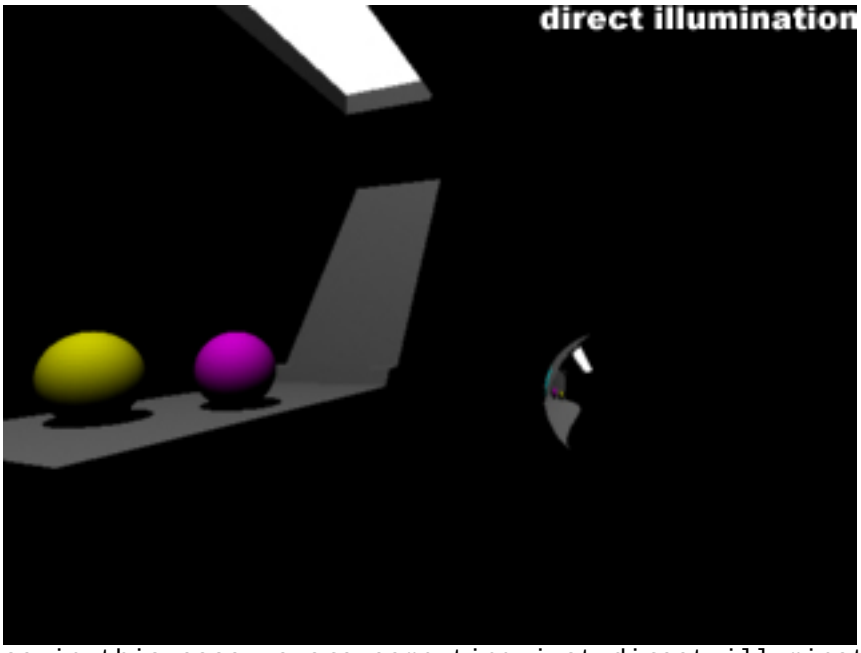
using lower values in this case produces spottiness, but it’s useful to explain a concept; imagine the photons being emitted by the light behind the distant door: most of them will hit the visible walls in front of the door, few of them are hitting the red sphere on the ground. Now the white spots represent the diffusion of the first ones and the red spots the diffusion of the second ones.



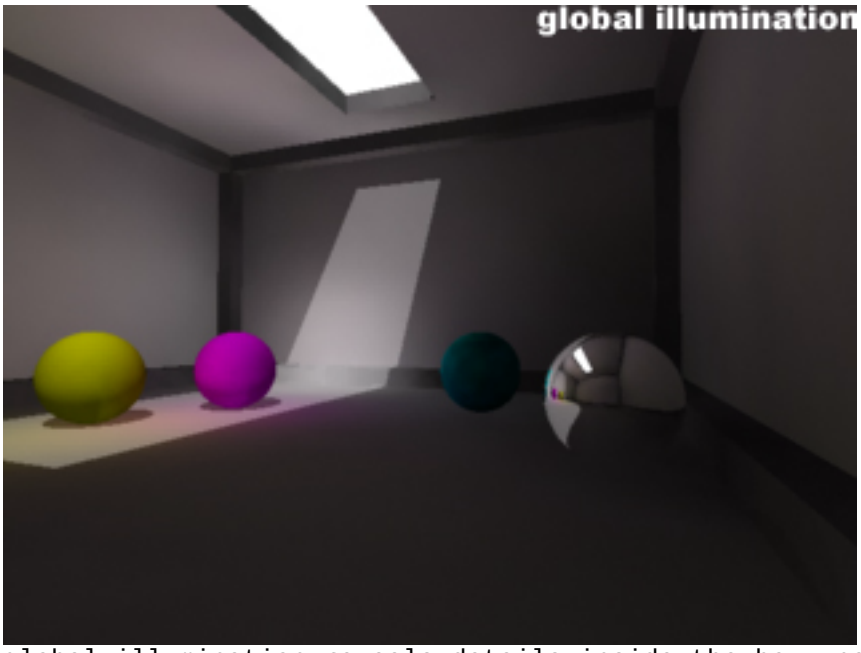
this is what happens when using higher values: illumination is computed on a too wide area and physical approximation gets lost.

- spheres in a top-windowed box, useful to show GI, color bleeding and the addition of a Final Gathering pass.

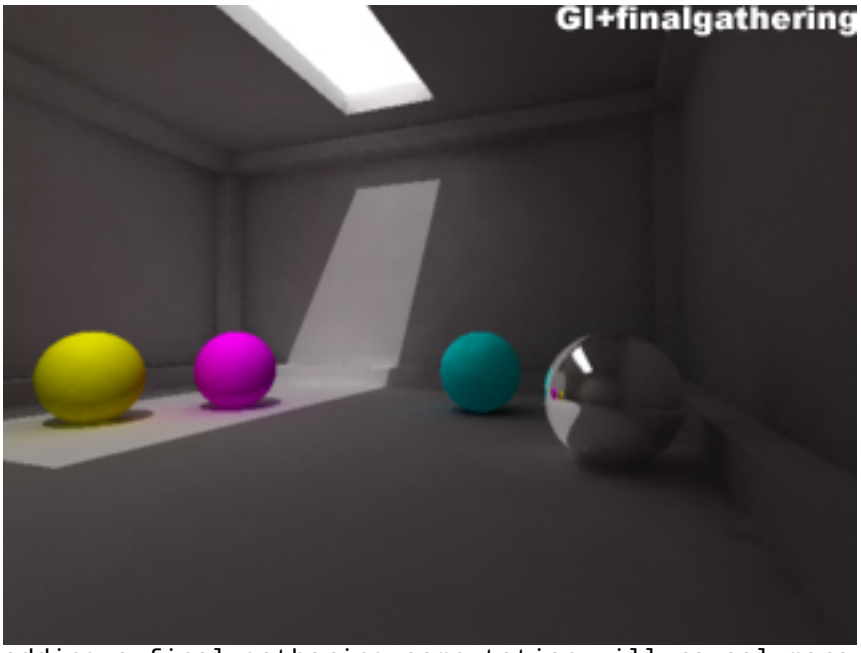
In this case we'll see a series of images showing first a direct illumination raytracing (with rt shadows) then the same scene rendered with global illumination and finally an image with the final gathering step and global illumination together.



so in this case we are computing just direct illumination, you can see raytraced shadows and a reflection of the chrome ball. We can obtain this render easily with the Maya renderer and with mental ray without using GI or pseudo-GI tricks.
Let’s see what happens using global illumination.



global illumination reveals details inside the box, now we can see the inner walls and the bluish sphere and a (forced) color bleeding on the ground, below the spheres.



adding a final gathering computation will reveal more details, due to its nature it adds a one-bounce-only indirect illumination, and now the illumination on the edges of the box structure, behind the chrome sphere and at the border of the ceiling window looks more correct. Refer to the video for a detailed explanation on workflow.

final gathering

mental ray for Maya supports natively another technique, this time useful for one-bounce only indirect illumination, the technique is called Final Gathering : it estimates illumination for a given point by either sampling a number of directions over the hemisphere over that point (such a set of sample is called a *final gather point*), or by averaging a number of nearby final gather points since final gather points are too expensive to compute for every illuminated point. This is done to compute the light energy contribution of the area surrounding every point, and only for 1st generation rays.

Hence, when illumination is computed on a diffusive point, a hemisphere above the point is computed for indirect illumination, then direct illumination is computed. Plus, if GI is activated then a contribution from the photon map is considered.
Final gathering can be used in many situations:

- when indirect illumination changes slowly in highly diffusive scenes
- to eliminates low-frequency noise when using GI with few photons
- together with global illumination allows more accurate physical coherency
- to reveal fine details

A quick note to remind that a “fast lookup” option is present in render globals. When activated, it stores the irradiance value with every photon, this requires less FG points because each photon is

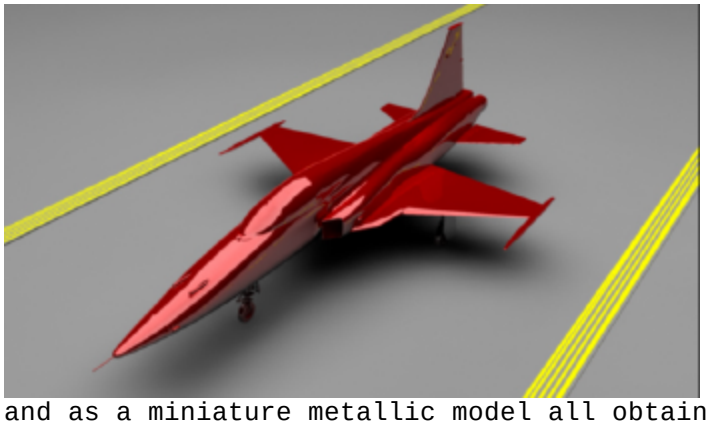
carrying a good estimate of local irradiance. Photon tracing is slower and uses more memory but rendering times is much faster. In this case remember to use less FG points.

We are taking a look at 2 scenes in the DVD:

- a plane rendered with no lights and no shadows
  Here I would first of all remind to turn off Maya's "default light" option in Maya render globals, otherwise at rendertime a directional light will be created.
  After having specified this, let's take a look at the rendered image:

This scene contains no lights and shadows - you're may wonder what's happening.
The scene consists of a plane (a free model downloaded at www.meshfactory.com), a bluish ground plane to cast the plane shadow and a white hemisphere over the plane useful to have the nice white reflection on the plane surface.
Light in this case comes from the shader of the white hemisphere which has an ambient component near white, it could also be incandescence.
Final gathering estimates illumination at every pixel by looking around every point being rendered, for example the "shadow" under the plane depends on the occlusion of the white hemisphere by the plane itself.
Final gathering works with every aspect of raytracing.
Shader attributes become very important to define the look of the rendering, we can control the irradiance estimated by final gathering via the attributes on a shader basis on every material shader under the mental ray tab via the "irradiance color" attribute.

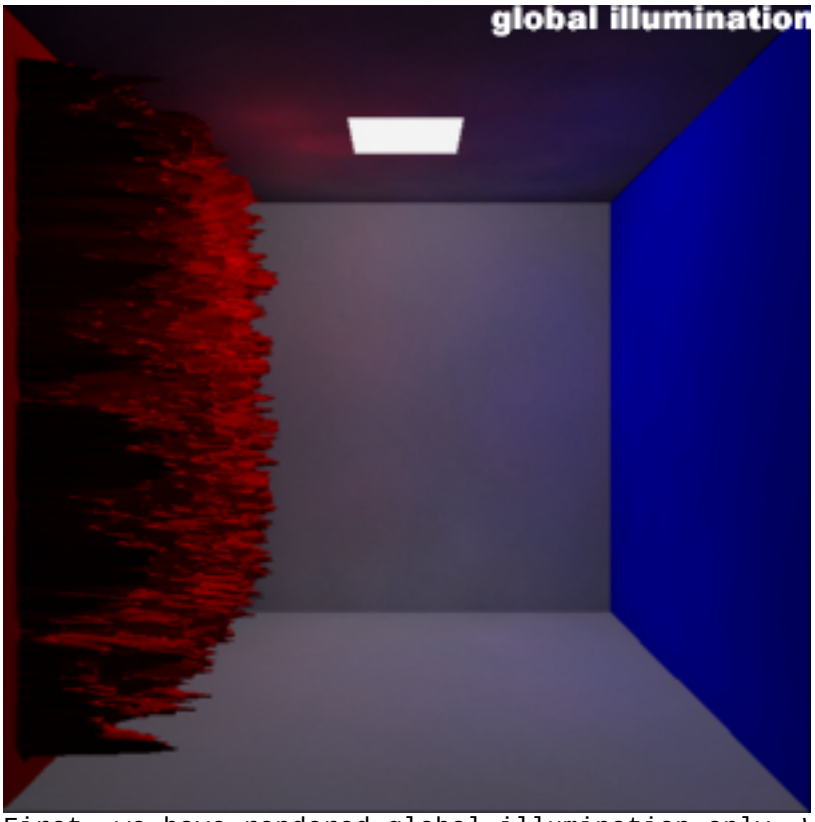
the same plane rendered with a softer irradiance on the ground plane (reducing the shadowing effect)



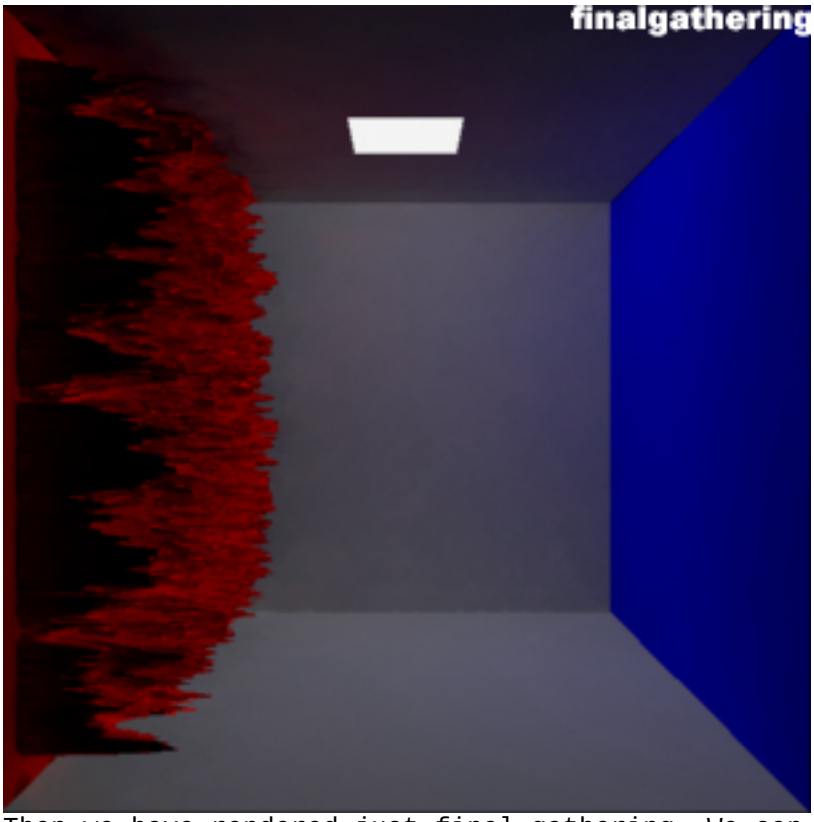
and as a miniature metallic model all obtained changing shader properties

- a Cornell Box with a displaced wall

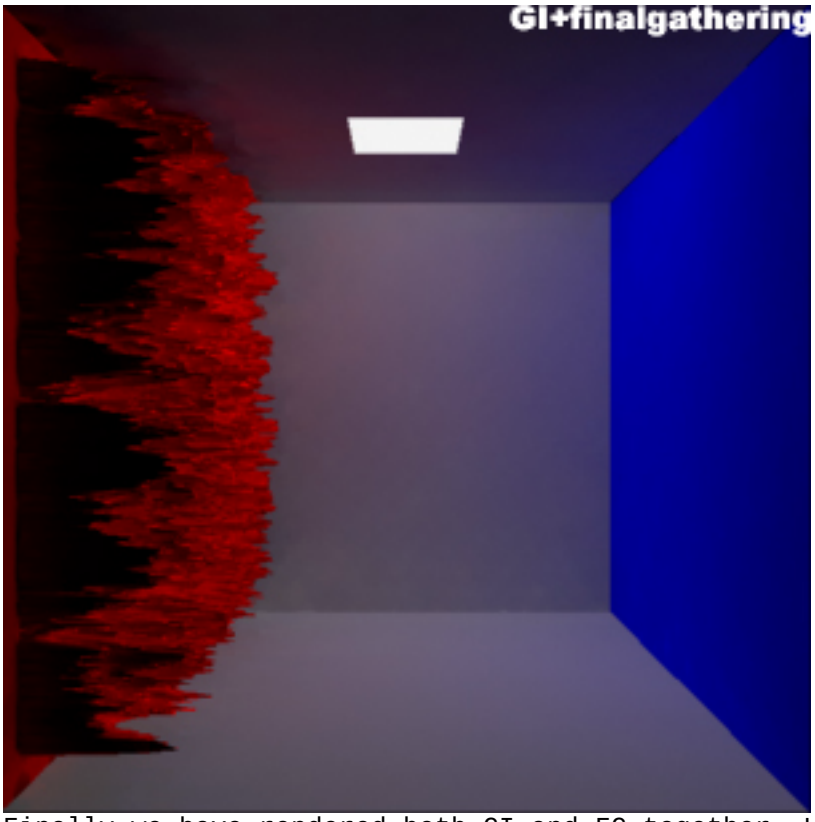
Final gathering and global illumination can be used together to achieve a more accurate lighting. (see also the global illumination chapter) For example, when fine details are in the surroundings and to reduce flickering in animations. When using FG together with GI you can reduce the number of GI photons and your energy statements and use less FG rays than when using only FG.
This scene features a Cornell Box with a procedural fractal fine displacement (see relative chapter) red wall and the whole scene is illuminated by a real area light.
The detail introduced with the displaced red wall is useful to understand when FG has a real utility combined with GI.



First, we have rendered global illumination only. We can see the color bleeding on the back wall and in general allover the image. Lighting between the displaced peaks on the left becomes darker at the bottom of the peaks



Then we have rendered just final gathering. We can notice a slightly different illumination in the displaced area at the left, lighting seems more accurate between the displaced peaks. On the other hand having the typical FG one-bounce-only indirect diffuse lighting isn’t so precise as the previous multi-bounce GI



Finally we have rendered both GI and FG together. Lighting now shows both correct GI approximation and good lighting between the peaks thanks to the addition of the one bounce only FG pass. To render this image we've followed the suggestions previously mentioned: energy has been reduced to 1/2, the number of photons to 1/10 precompute photon lookup is set to ON.
As a result, we have a more accurate and detailed illumination all over the peaks and depressions of the displaced wall and on the corner between the walls of the box.

nonzero area lights

Maya's area light simulates diffuse light as a real area light, while for the specular component it uses an internal approximation which provides a quick simulated behaviour of a real area light.

mental ray for Maya translates the Maya area lights correctly and in addition provides real non-zero area lights with sampling control of the primary and secondary rays (low sampling). Real area lights are available via Maya point and spot lights on the arealight tab present in the lightshape AE. There are 4 primitives supported: rectangle, disc, cylinder and sphere, all defined by a 2D directional sampling (these surfaces can in fact be all defined by 2 vectors).

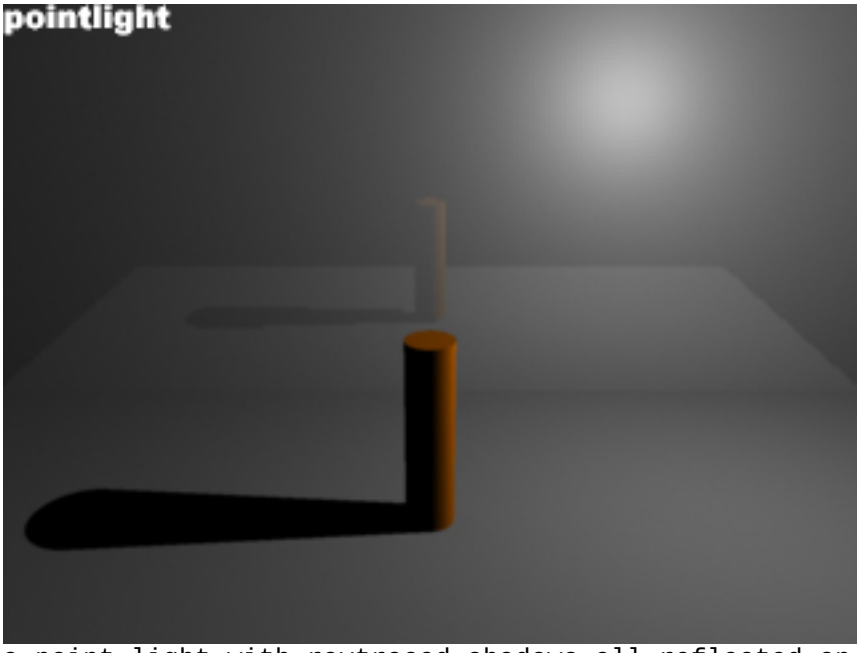
You can access real non-zero area lights in every light AE under the mental ray tab. After turning 'on' the area light, you can choose

which geometric type, low sampling and visibility. To change the size of your light, use the scale manipulator on the OpenGL view.

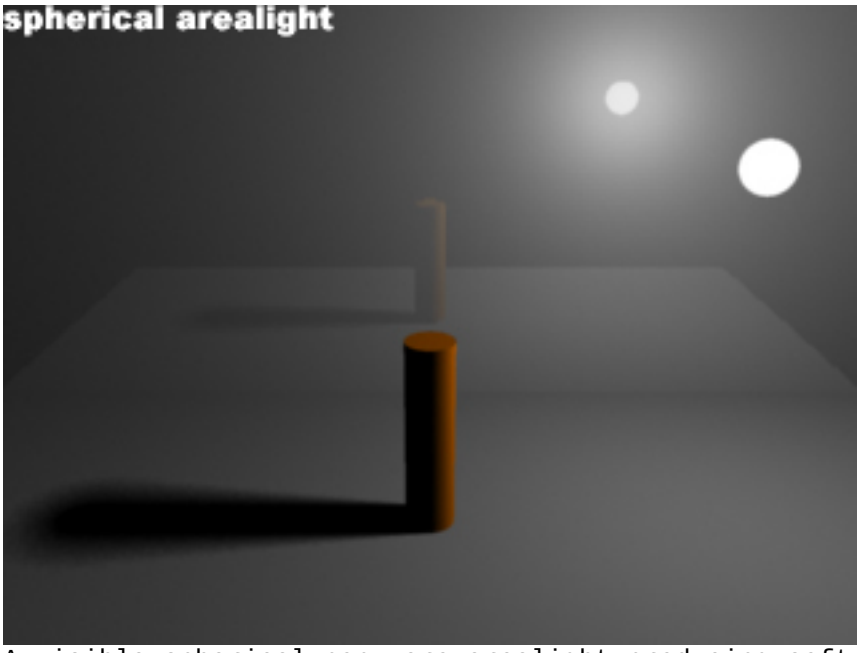
When using mr area lights, the light radius parameter should be 0 to avoid artifacts.

As you can see in the DVD, the main purpose of area light sources is to generate more realistic lighting, and usually results in:

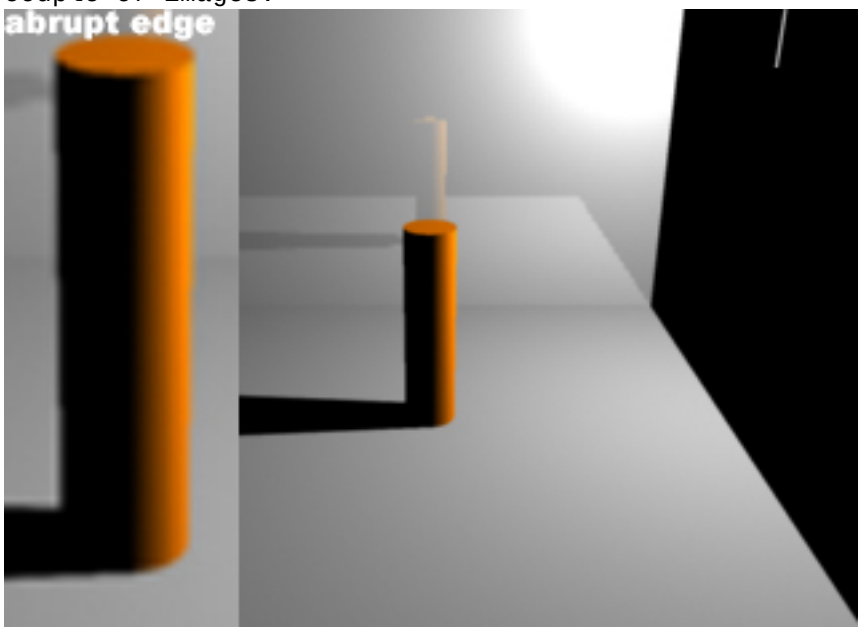
- soft shadows (a point on an object may be illuminated by only a part of a light source producing then a soft shadow, analog to the penumbra produced by an eclipse) the two following images to compare a standard pointlight to a non-zero arealight



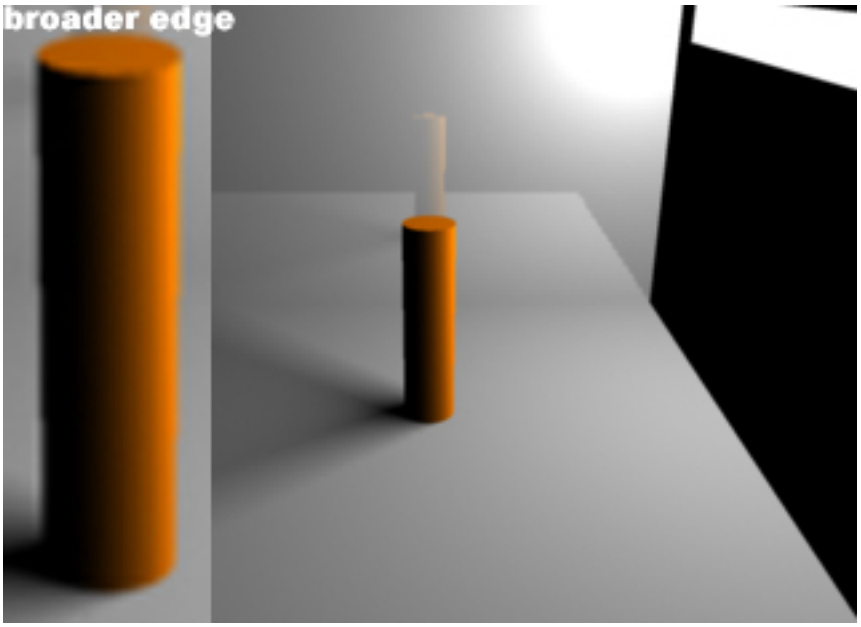
a point light with raytraced shadows all reflected on a neighbor mirror.



A visible spherical non-zero arealight producing soft raytraced shadows reflected on a neighbor mirror.
mental ray 3.1 supports also ‘user’ and ‘geometric object’ area light, as for now these two features are available only via custom shaders.

- Broader edge lighting, meaning that light wraps more around the object for the same reason, as we can easily see from the next couple of images.
- 

The thin white line you can see in the top-right part of the image is a small area light. Zoomed on the left, the illumination on the orange cylinder, note the hard edge shadow.

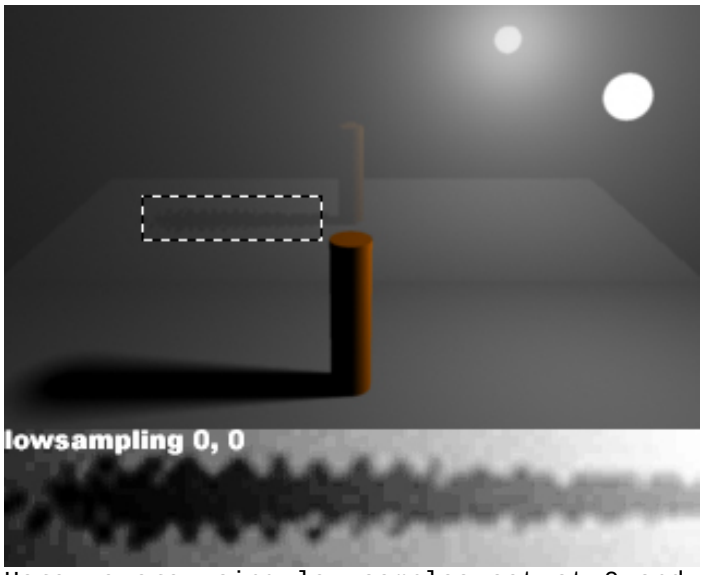


Here the area light has been scaled up (you have feedback in the UI) in order to show the broader edge of the lighting on the cylinder and the relative soft shadow.

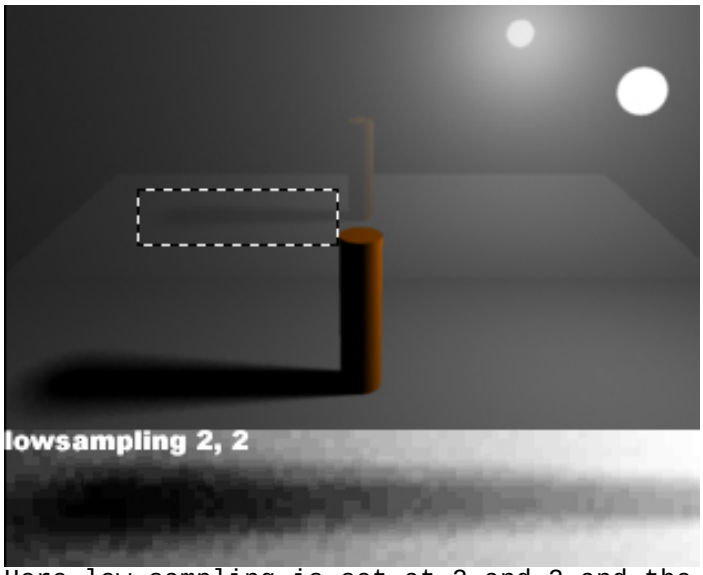
I’ve been placing a mirror behind the cylinder to show another interesting characteristic:

Sampling over the light surface can be controlled via the sampling attribute, while for 2nd generation rays (eg the reflection of an area light on a neighbour mirror or refractions) it can be controlled via the low level value: the light will sample with the low sampling values if their sum is superior to the low level value. As default, reflections/refractions are sampled at 2 and 2. Ray depth should be higher than one for a light shadow to be reflected.

The next couple of images show what happens on the reflections when using low or high samples rate for our area light:



Here we are using low samples set at 0 and 0 consequently on the mirror, the shadow results are very imprecise as you can see from the zoomed region while on the plane the shadow is sampled with the renderglobal settings

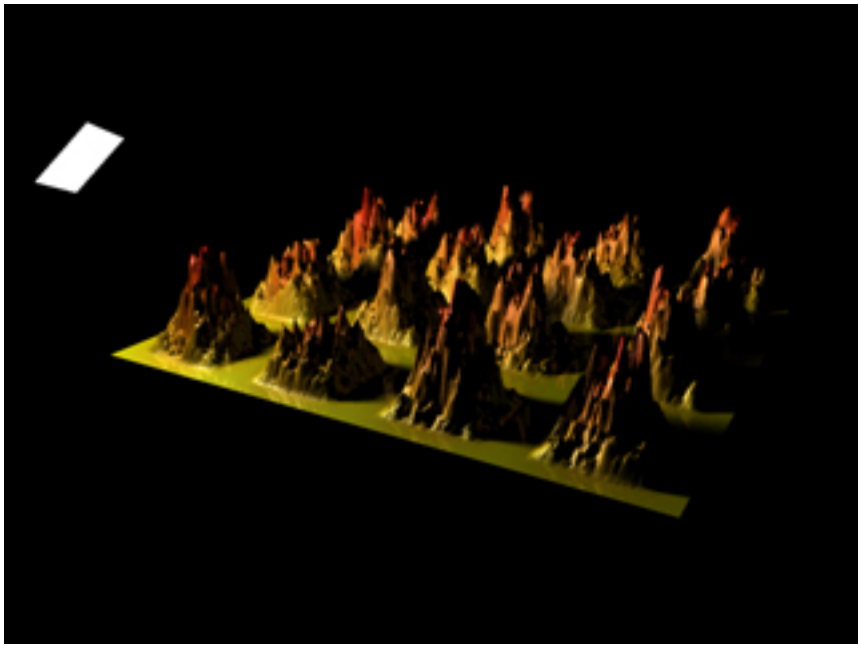


Here low sampling is set at 2 and 2 and the shadow on the reflection results are much more accurate.

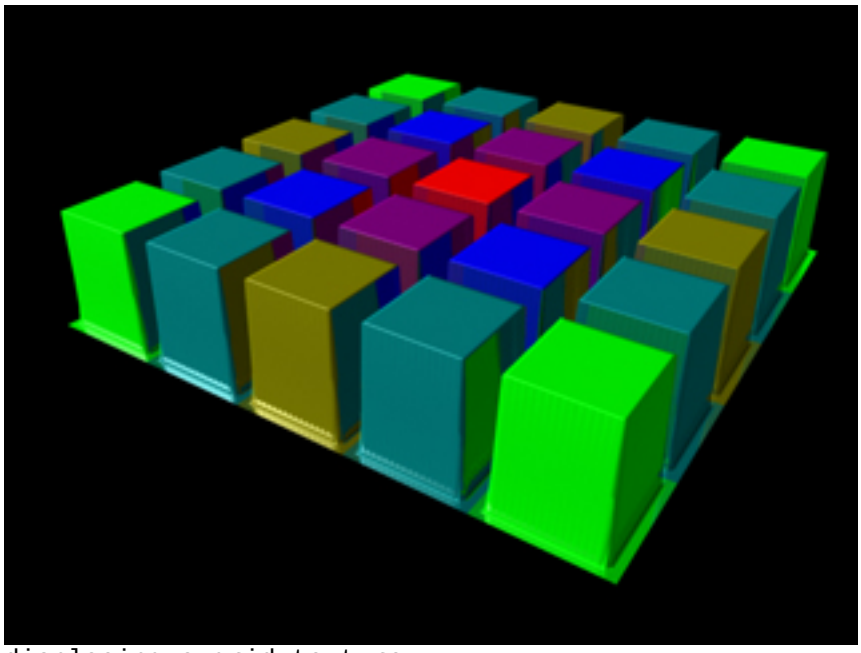
fine displacement approximation

Displacement approximation in mental ray for Maya by default is ‘derived from Maya’ using a technique analogous to Maya’s feature-based displacement mapping.

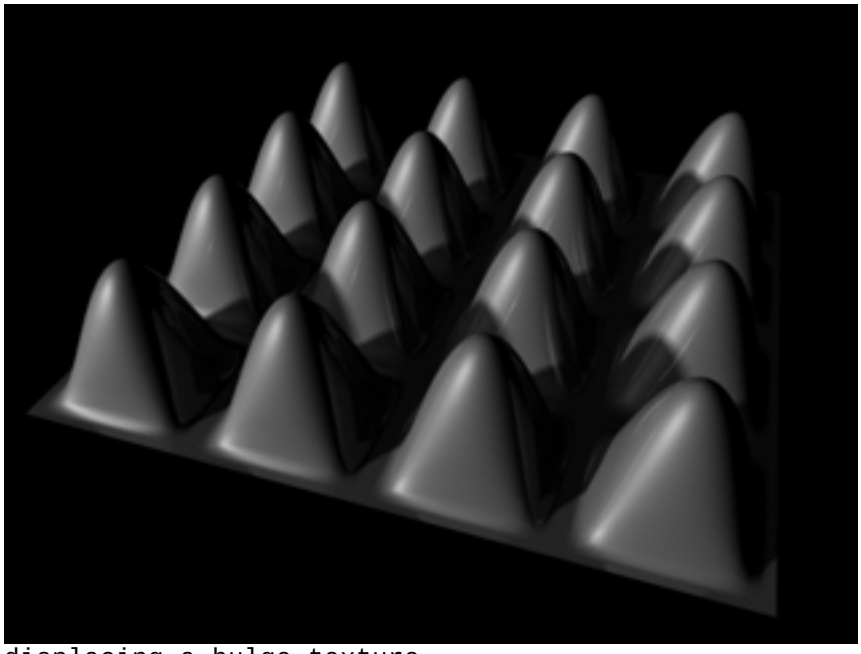
There’s a new subpixel displacement technique called “fine” displacement (coherently with mental ray 3.1 specification). It’s a tessellation technique which can be used for surface and displacement approximation and is fully integrated with every aspect of raytracing. The following image features fine displacement with real non-zero area lights producing raytraced shadows and reflections.



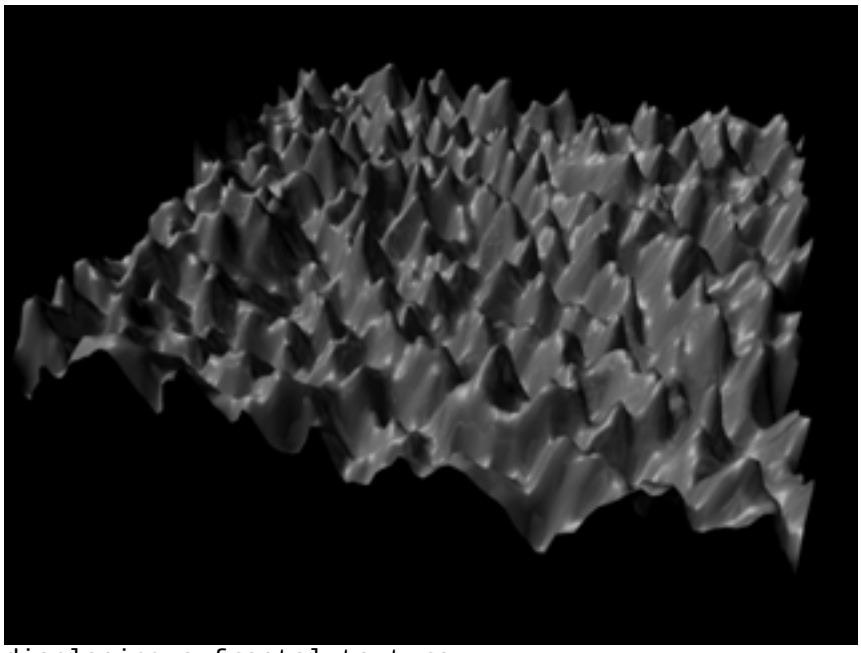
As output you’ll get a very fine microtriangle approximation that works especially well for very detailed displacement maps, resolving every little detail instead of introducing tessellation artifacts such as broken edges at contrast boundaries in the displacement map. The algorithm performs very well both for abrupt, smooth and noisy displacements as you can see in the following series of images:



displacing a grid texture



displacing a bulge texture



displacing a fractal texture

In mrfM you can access the approximation style via:

Window> rendering editors> mental ray> approximation editor

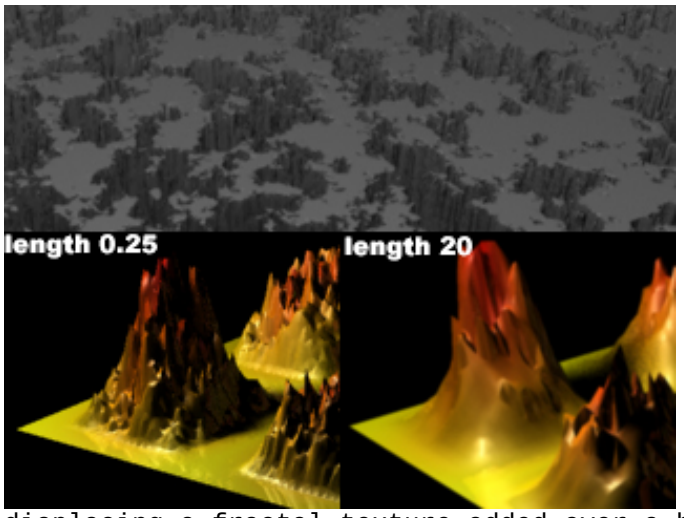
Select the object shape you want to apply the approximation style to and on the Approximation Editor click on the button ‘create’ and then ‘assign’. A new node called mentarayDisplaceApprox (or mentaraySurfaceApprox) will be connected to your object shape (if you want all your objects to be rendered with this approximation, go to render globals> yourRenderQuality> overrides> tessellation).
When you click on the ‘edit’ button or select the node you’ll see the relative AE with all the attributes: ‘fine’ approximation style is only available with the ‘Spatial’ approximation method.
You can set the following attributes: Min and Max Subdivisions, Length (view or non-view dependent) and Sharp.

View: displacement is now working in in raster space, interpreting the parameters in pixels instead of 3D units. Effectively, closer objects are tessellated more finely (take in mind if instanced objects are present they may be triangulated multiple times).

Length: specifies the minimum triangle edge length under which tessellation is no longer performed.

Min and Max subdiv: is a recursive subdivision process, it can be controlled setting these two boundary levels.

The following image shows the difference in rendering with different length parameters using a fractal texture (over) as displacement. Using length values lower than zero results in subpixel displacement.



displacing a fractal texture added over a bulge texture

A final note on memory. Fine displacement critically depends on the specification of a cache size limit. Otherwise, the fine tessellation results would not flow through the cache but accumulate until memory runs out. Physical and virtual memory can be controlled under:

mental rayglobals> memory and performance> memory limits

-1 means default, for physical memory stands for 512MB of RAM,

There are 2 fields: physical and virtual memory, physical memory is important for fine displacement performances: the default value, -1, stands for 512MB of RAM, lowering this value too much could result in a continual flushing of the cached chunks created by the fine displacement (you can eventually see this activating verbosity during rendering). 0 value means infinite, and you can specify your own value which should be the half of your physical RAM installed.
You can override this value in the UI as just explained or with the –jobmemory option using mental ray standalone.

Virtual memory is useful mostly for .map textures. These are memory mapped textures equivalent to Maya BOT textures. Values have the same meaning, the default -1 stands for 1GB of disk space
.map textures can be generated via the command line utility imf_disp (in your mrfM/bin directory), you can actually convert a texture eg from .tif to .map keeping the same extension (in this case tif), hence, you won’t need to change the reference path from your filetexture node.

HDRI techniques

mental ray for Maya 1.5 now supports high dynamic range textures as input and output (HDR .hdr, mental ray HDR .cth and various floating point formats).

You've probably seen many related tutorials published all over the internet. The website of Paul Debevec www.debevec.org , who I want to thank for letting me use his HDR probe textures for this DVD, will answer all your questions on what are HDR textures, how to produce and practically use them. He has an impressive collection of related papers presented at siggraph you definitely have to look at.

In any case, few words are worth here: the HDR acronym means the higher dynamic range of colors resulting in a wider range of pixel color values stored in a texture. The "dynamic range" of a scene can be represented as the contrast ratio between its brightest and darkest parts. HDR images can be created taking multiple pictures of the same camera view (scene) at different exposures in the real world or as an output of a capable renderer.

So, if in a usual .iff texture you have the highest value of RGB set to 1,1,1 (normalized white, which would be 255,255,255 in the 0-255 range of a 8-bit per channel image) in a hdr texture you will have 'super-whites', like 8,8,8 or 16,16,16 and in general all other 'super' colors. In practice, these high dynamic range pixels are represented by floating point numbers and unlike most regular images whose pixel values are nonlinearly encoded, pixel values are proportional to the amount of light in the world corresponding to that pixel.

In the end for rendering software, colors are numbers and what is really important is that this whole new range of colors becomes a wider range of values which can be used by algorithms like final gathering and raytracing to compute illumination, reflections, refractions etc.

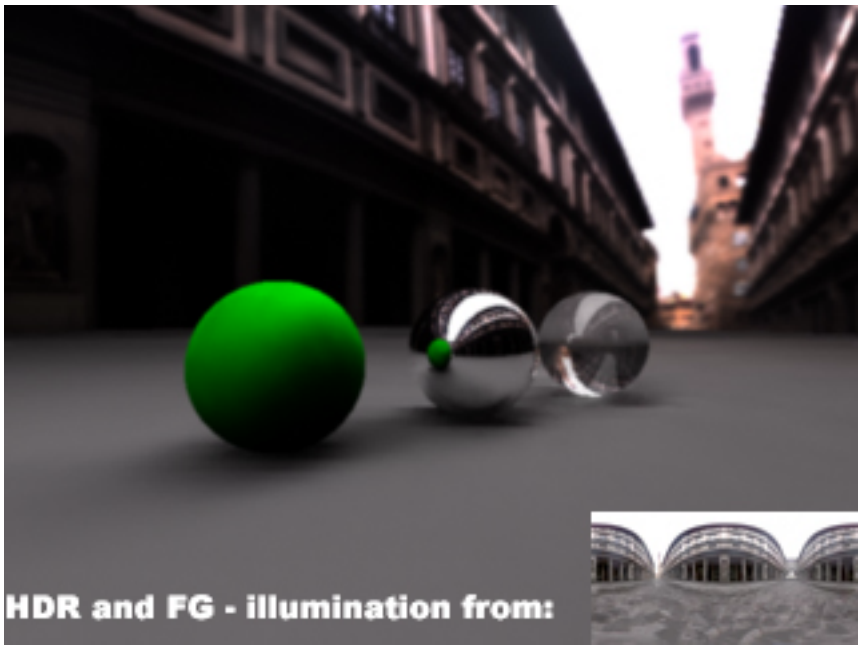
We are going to see:

- HDRI for illumination,
- HDRI for reflections and refractions
- HDR output in mental ray

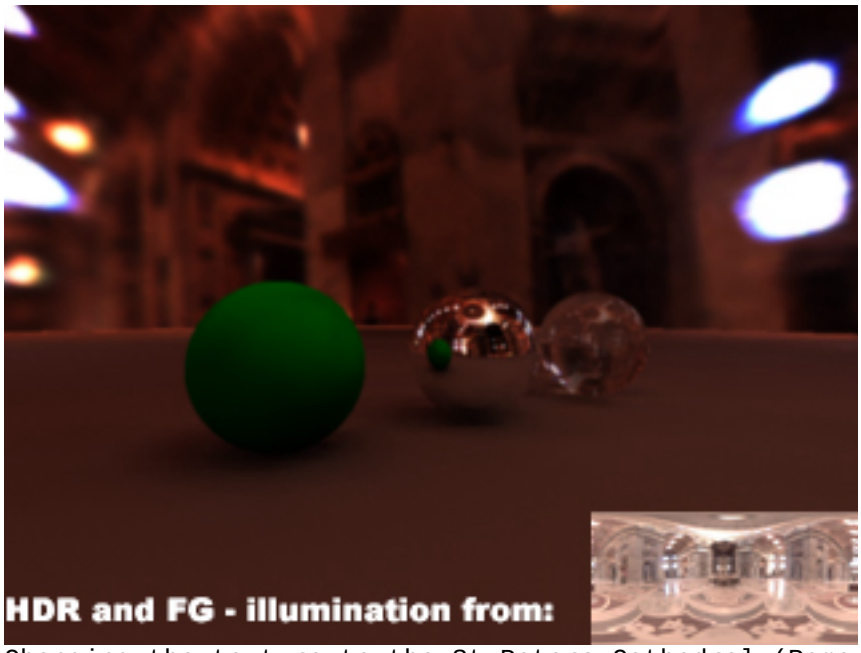
Before starting a quick note: when selecting an HDR image file to use as a file texture, this image will not appear in file swatches or in the hardware render or when rendering with the Maya Software renderer; it only works when rendering with the mental ray renderer. You'll also get a warning in the script editor, just don't bother with it.

- HDRI for illumination

One of the most common ways to use HDR textures is to let them define the enviroment illumination of a scene thereby leaving mental ray's final gathering algorithm to compute the illumination levels.



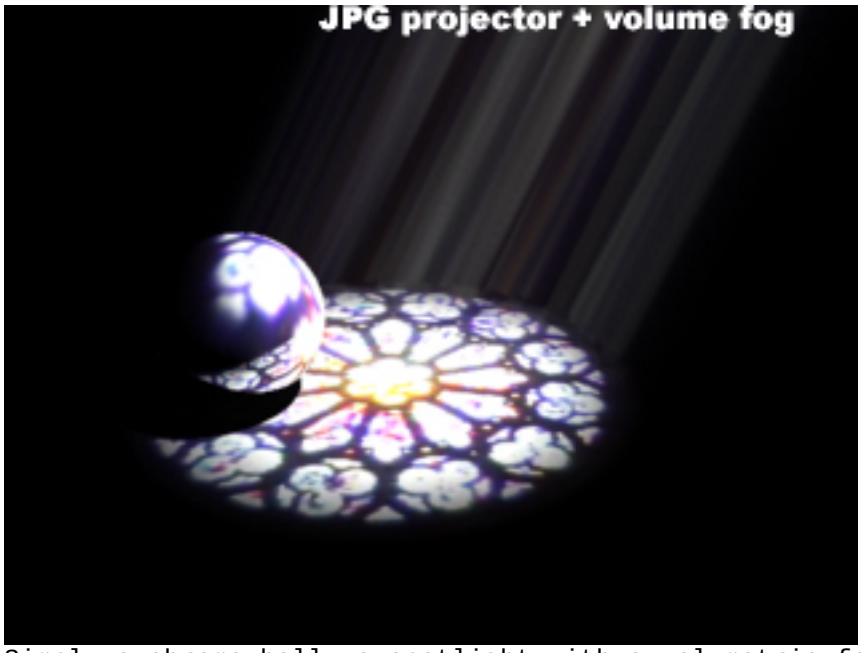
In this very simple scene an environment sphere texture is wrapped around the scene, and the hdr texture is piped in to the file texture node as color, then ambient or incandescence is set to non-black.
In this way, final gathering picks up the dynamic range of color for illuminating the scene. The HDR texture is the Uffizi in Florence, Italy. Notice how the white color of the sky gets nicely and dynamically readapted on the reflective color of the chrome sphere (100% reflective) and the glass sphere (20% reflective).
See later for HDR reflections and refractions.



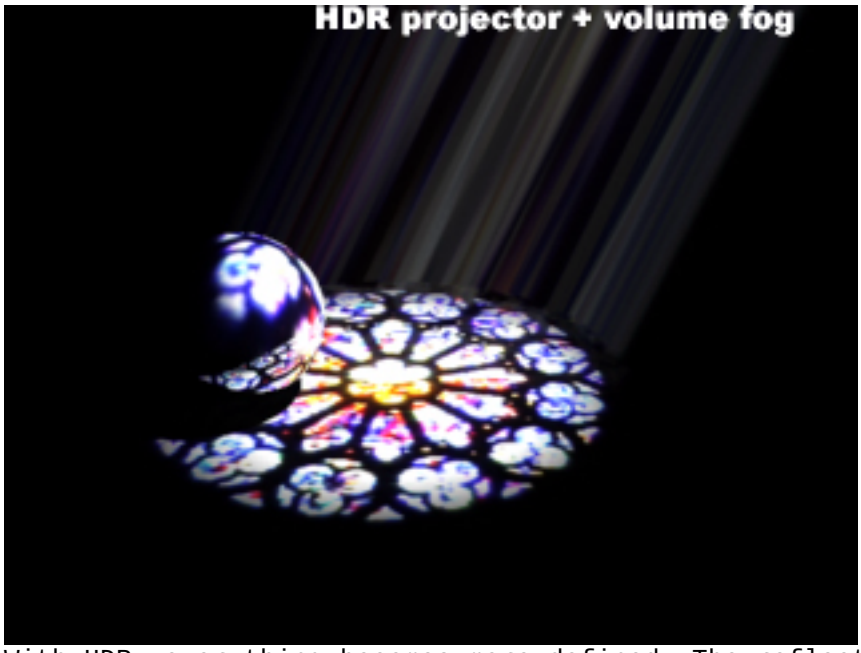
Changing the texture to the St.Peters Cathedral (Rome, Italy) in the same simple scene results in a change in the illumination. To control or change the color of the illumination you can change the intensity

via the color gain attribute in the texture node and on a shader basis you can always control the irradiance with the new ‘irradiance color’ attribute in the mental ray tab of your material shader’s Attribute Editor.

Another way to use your HDR texture for illumination is for example to map the color of a spotlight with an HDR file texture, what we could call a light projector. I’ve prepared a couple of images first using a standard JPG texture and then using HDR to see the difference.



Simply a chrome ball, a spotlight with a volumetric fog node (remember to override the volumetric sample in the cone shape render stats) connected and a JPG texture mapped as light color. Notice the blurriness caused by the JPG clamped color range.



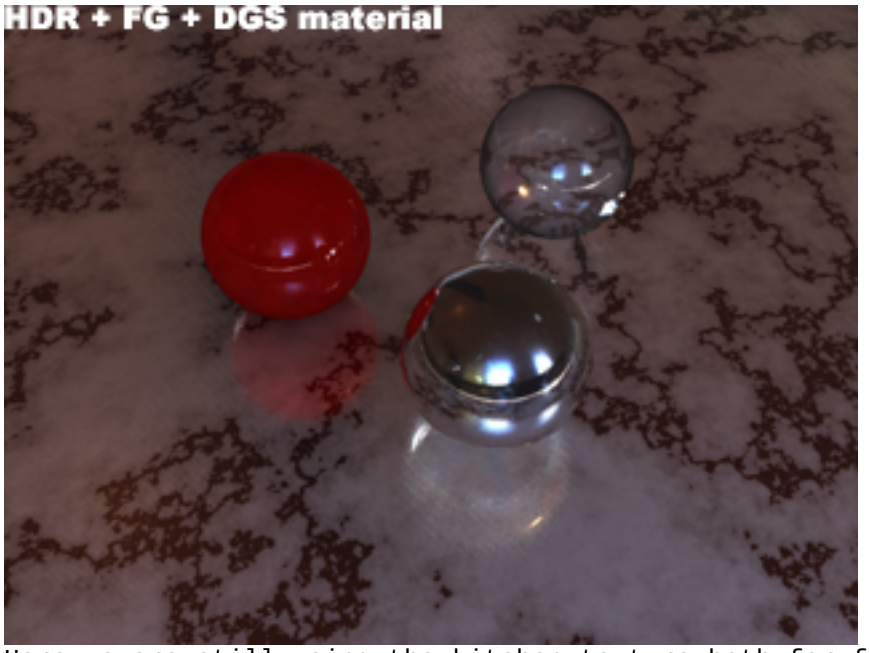
With HDR, everything becomes more defined. The reflection on the chrome sphere and the projection of the light, thanks to the unclamped range of color. As you can see, the volumetric rendering of the Maya foggy spotlight gets correctly rendered by mental ray's raymarching algorithm.

HDRI for reflections and refractions

When you want to use your HDR texture for reflections/refractions you need to map your texture to the "reflected color" attribute of your specular-component shader. Usually, with more objects in the scene you should use an environment texture to map on the "reflected color". With the values unclamped it will allow a more accurate adaptation on the reflections of different specular materials in use.

This image uses Paul Debevec's Kitchen texture mapped as reflected color on both spheres and on the ground.
See the different adaptation of the reflection first on a plastic shader (the mental ray billiard ball) and then a chrome shader which features more reflectivity and has a 100% white specular color. The higher dynamic range allows the plastic shader to show the reflection of the tree outside the window while the reflection gets blown on the chrome shader because of the characteristics of the shader itself.
Final gathering is also activated to compute illumination (no lights in the scene).

Following is another image with different materials.



Here we are still using the kitchen texture both for final gathering and reflection/refractions, plus the floor is now a marble DGS material. As you can see, the reflections on the floor becomes softly blurred with distance, see later on the extra section for the blurred reflections via DGS material.

- HDR output in mental ray

First of all: the output of HDR images cannot work in the UI render (render view), you need to use instead the batchrender.
As long as you know this, the correct way to proceed is to create a new framebuffer option in:

MentalrayGlobals> format&resolution

A click on the map icon will create a new framebuffer, now to output HDR you need to use:

output format: HDR
datatype: RGBE (byte) 4x8 bit

Which is a special mental ray frame buffer which stores RGB colors (no alpha) as floating-point values combining the storage efficiency of 8-bit RGB with the ability to store values greater than 1.0.

You can also output pure floating point formats using for example TIFF and RGBA (float) 4x32 bit but your file will be 4 times bigger in size.

Finally, to see your image (which will be rendered on the mental ray directory of your project) use the mental images utility imf_disp in the /bin directory of mrfM.

Light baking

With version 1.5 of mental ray for Maya, we can now access the “convert to file texture” function via the “mental ray baking options” available in the hypershade window:

Hypershade: Edit> Convert to file texture (mental ray)

As you can see from the option window (and in the online documentation) this implementation supports both direct and indirect illumination (which can be baked independently), shading and shadows and it’s based on the “lightmap” rendering function available from mental ray 3.0.

Two main ways of light baking are available:

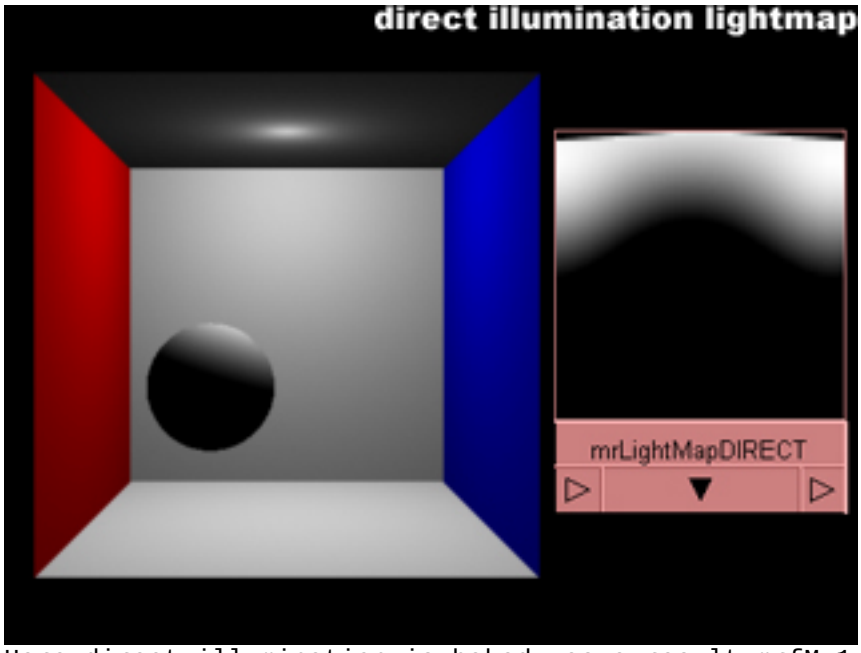
- To texture
- To polygon vertices (per vertex color)

In general terms, this is a method for sampling an object before rendering and storing the results for later use.

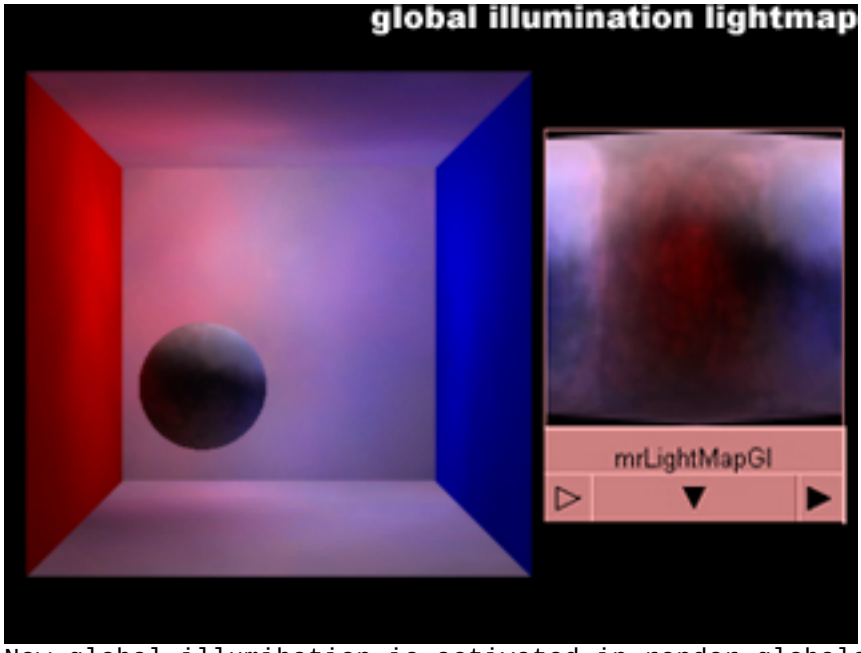
The most common application samples illumination for each point of a texture image wrapped around the object, and stores that illumination in the texture. The sampling can be controlled for texture by the UV texture space and also on the vertices of polygon meshes.

Illumination is effectively frozen into the texture, which can later be mapped onto an object in the conventional way.
The advantage is that rendering can obtain illumination quickly from the frozen illumination, instead of computing it at rendering time. This is especially valuable for indirect illumination, which takes more time to compute than direct illumination.

As previously said we can decide to bake using various options, for example direct or indirect light (GI and FG); in the following images we have a Cornell Box scene setup with a floating sphere in the middle on which various options of illumination were baked.

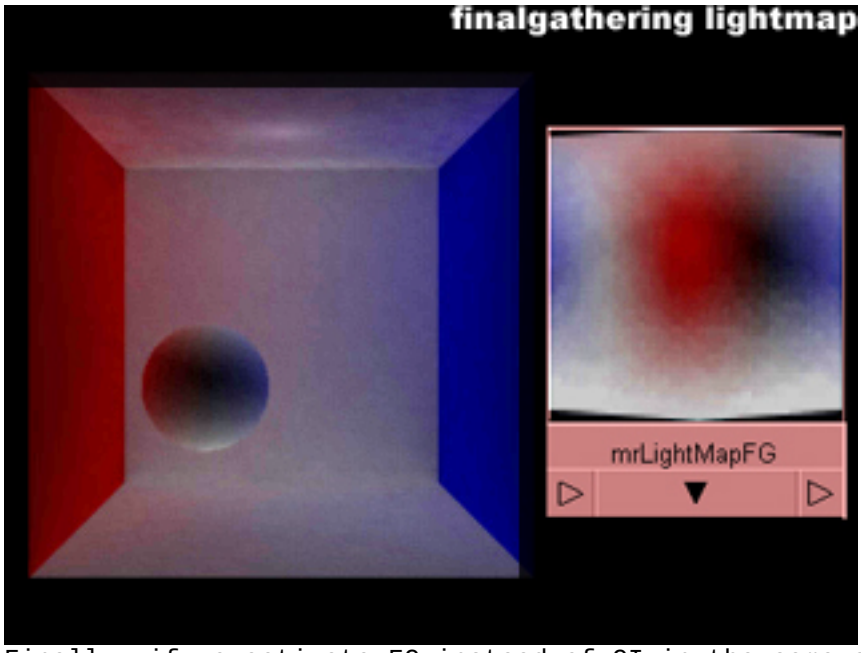


Here direct illumination is baked, as a result mrfM 1.5 writes a texture (you can follow the process on Maya help line if at least “progress messages” are selected on the export verbosity field), it’s rather obvious the white part of the texture represents the sphere surface facing the light. In this scene there is no global illumination nor final gathering activated and we are baking “incoming illumination only”.



Now global illumibation is activated in render globals and we are emitting GI photons from a point light inside the Cornell Box.
In the baking options “incoming global illumination only” is chosen. On the texture written by the lightmap rendering we can clearly see the

diffused indirect illumination caused by the photons bouncing first on a colored wall and then on the white sphere, storing on it the transported color.



Finally, if we activate FG instead of GI in the same scene we can bake its one-bounce-only indirect illumination using the same option “incoming GI only”.

In the DVD you can see how to bake your illumination to vertices and have an instant feedback on the OpenGl viewports and double checking color values using the component editor on selected group of vertices. Here I prefer to show the possibility to render the global illumination only (so this is not anymore related to light baking but it can be used to obtain “light passes” to composite later on, in order to control illumination independently) which is now possible with the final release of mrfM 1.5.

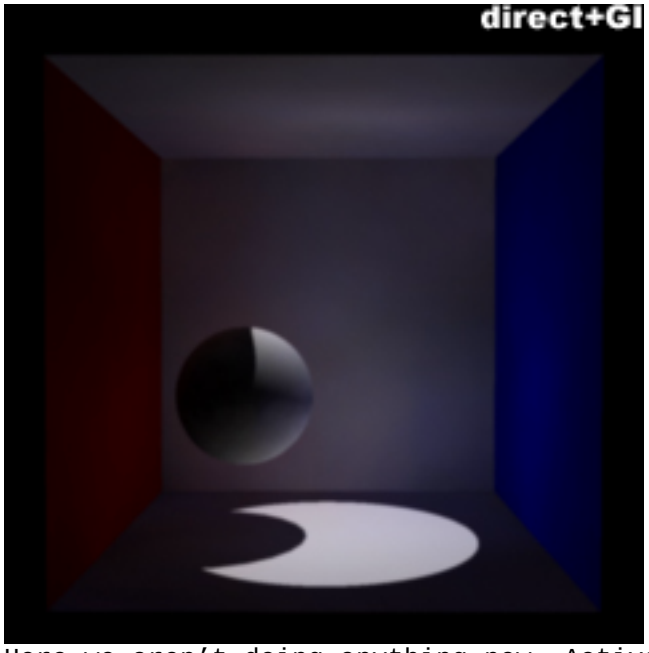
After you’ve set up the scene to obtain GI, this is possible performing two actions, first you need to activate the “render global illumination global pass” flag on:

mental rayGlobals> quality> render globillum global pass

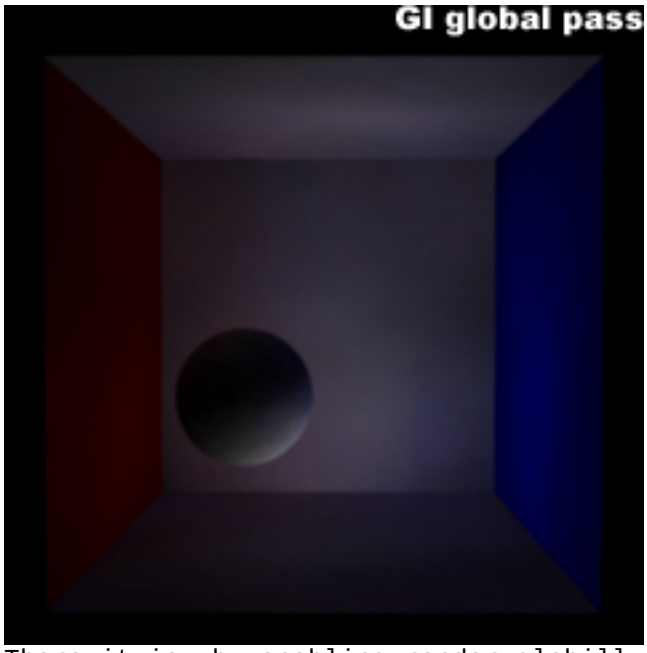
and then activating global render passes in the Maya render globals together with disabling *all* global render passes (setting them to off) in the lower table:

Maya render globals> render layer/pass control> enable global passes

This is the same Cornell Box setup with a spotlight instead of a pointlight, it’s rendered in the renderview as a standard rendering in direct illumination.



Here we aren’t doing anything new. Activating GI we can better see the scene environment because in addition to direct illumination photons are revealing surroundings.



There it is, by enabling render globillum global pass in mental ray globals and enabling render passes in Maya render globals (switching all passes to off) we get basically the difference between the previous images, meaning: “direct+GI” - “direct only”= GI only.

Some considerations on light baking:

- Once a light map is written to disk, it becomes just like any other texture and so it can be eventually modified with any texture editing software. So, manipulating the texture maps can be useful to obtain/correct lighting effects in a free and fast way.

- Lightmap textures can be mapped to any shader “irradiance” attribute in order not to recomputed indirect lighting in successive renderings. In this case the “irradiance color” sets the “strength” of the irradiance cached map.

- The baking process is performed at the quality settings in mental ray globals.

- motion blur effects may not appear, blur your texture maps with a filtering option in a compositing software instead/

- volumes are not considered.

- the alpha channel is always rendered in file textures.

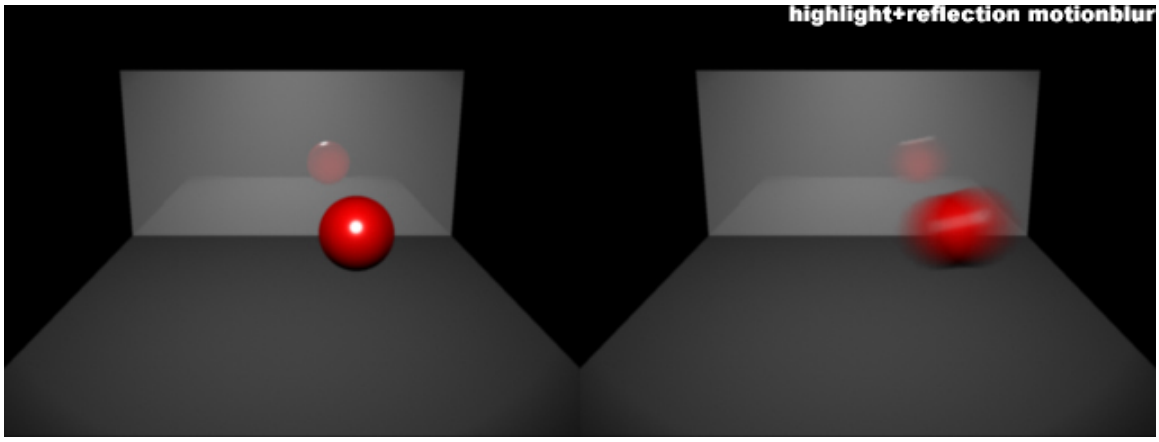
- Baking object “A” and then baking object “B” is different than baking them together.
  baking(A) + baking(B) != baking (A+B)
  It’s more noticeable when using indirect illumination. This is because once an object gets baked it’s assigned to a surface shader and no longer reflects light in the scene.

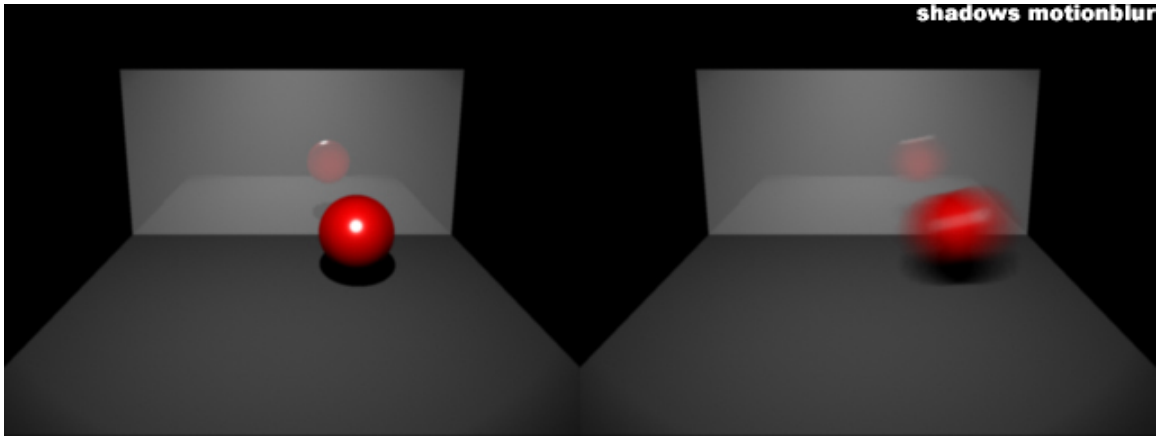
motion blur

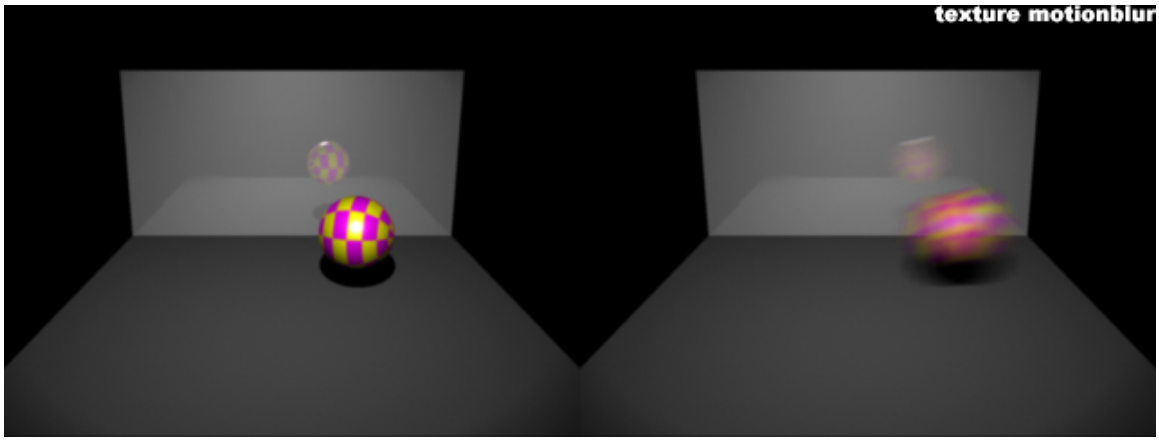
In mental ray for Maya the motion blur algorithm works with scanline, raytracing and indirect illumination effects. It basically motion blurs everything: shaders, highlights, textures, lights, shadows, cameras, caustics, transparency, intersecting objects, reflections and refractions. Of course motion blur introduces rendering overhead, but only where blur must be computed. If the camera moves this could be the entire image but if only a small object moves the overhead is small.

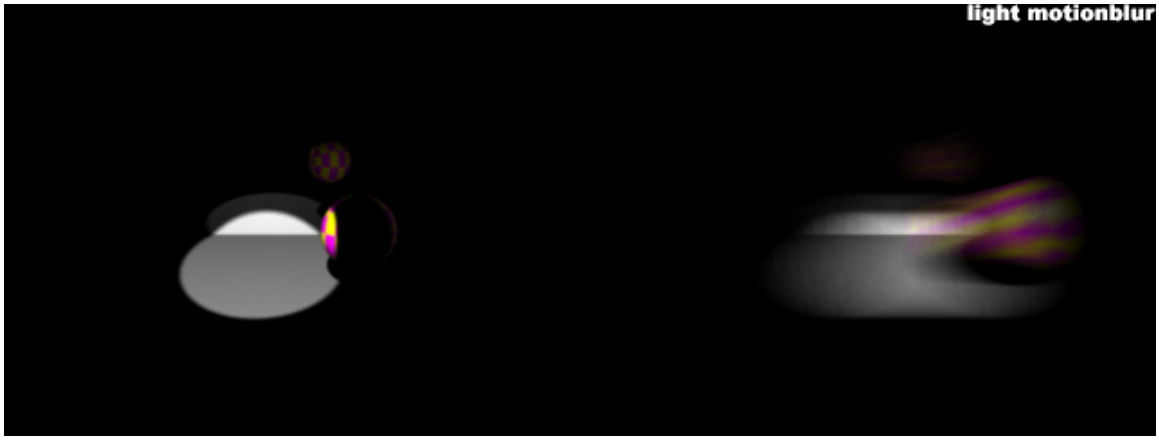
Motionblur can be activated in mental rayGlobals using the “previewMotionblur” preset or manually in your user-created render quality setting. Options can be defined under the motion blur tab.

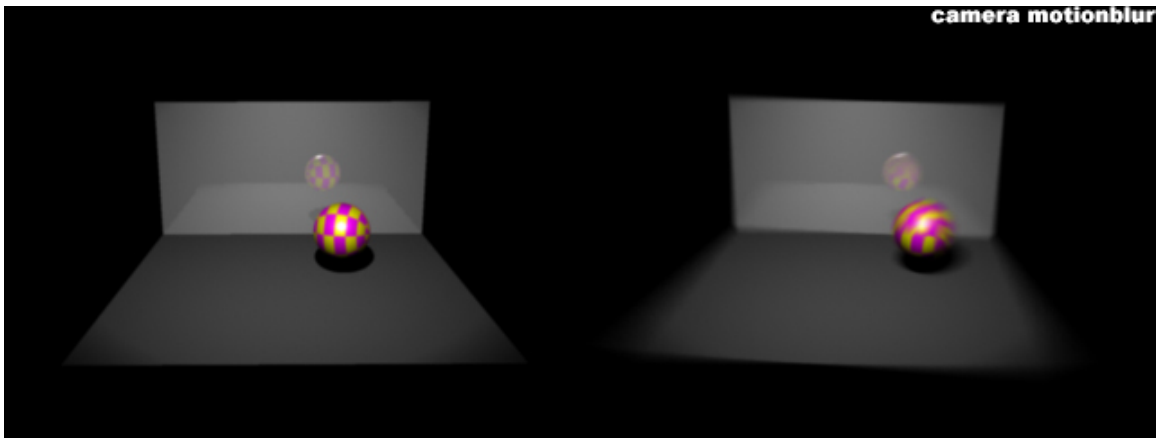
All the previously mentioned features are shown in the following serie of images based on a setup scene with a sphere bouncing on a plane (with an impulse coming from left to the right). There is also a back mirror to show the capability of motion blur to work on secondary rays. The left image is not motion blurred, the right one yes. Features are reported on image basis.

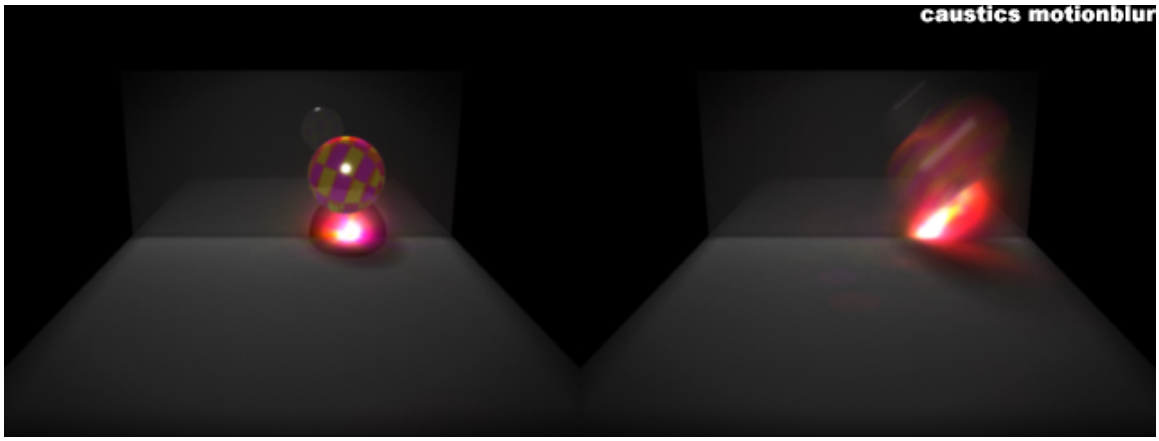


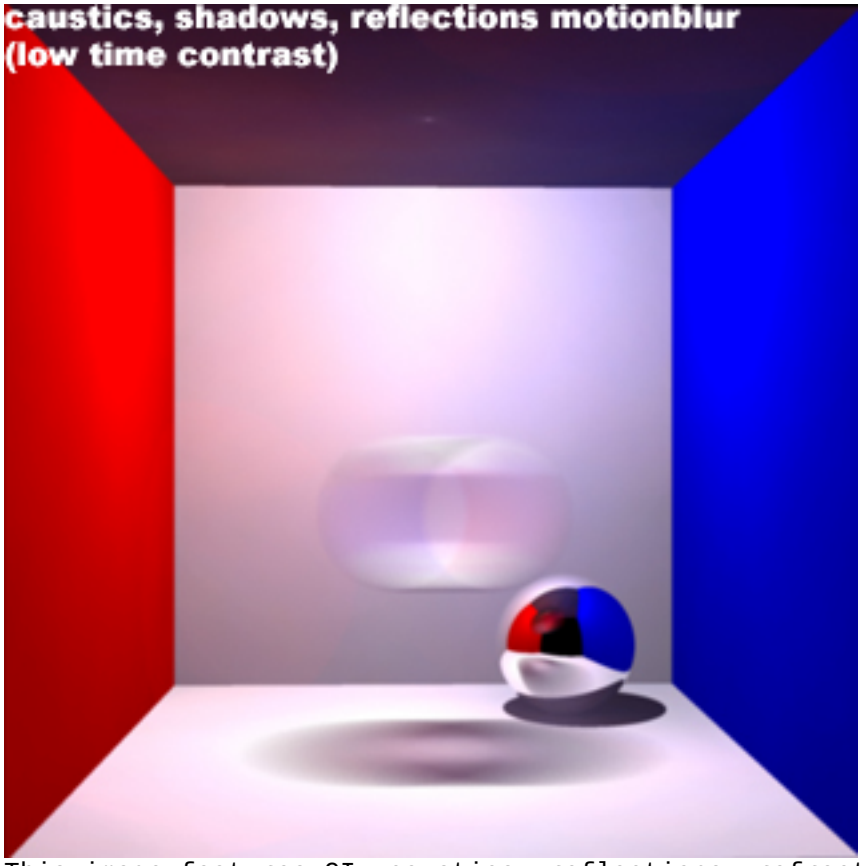



texture motionblur


light motionblur


camera motionblur





This image features GI, caustics, reflections, refractions and shadows and it’s rendered with low time contrast settings in render globals to obtain high quality motion blur.

As specified in the mental ray online reference manual, there are two alternative motion blur algorithms.

The regular motion blur algorithm is based on adaptive temporal over sampling. It’s enabled if the shutter period is nonzero for example when the shutter time is greater than the shutter
delay time.

The fast motion blur mode is based on coordinated spatial

and temporal over sampling, which requires far fewer samples (typically by a factor of five) for motion blurred pixels. Fast motion blur mode is enabled if the shutter period is nonzero, and if the time contrast is set to 0 and if the min and max sampling values are set to the same value.

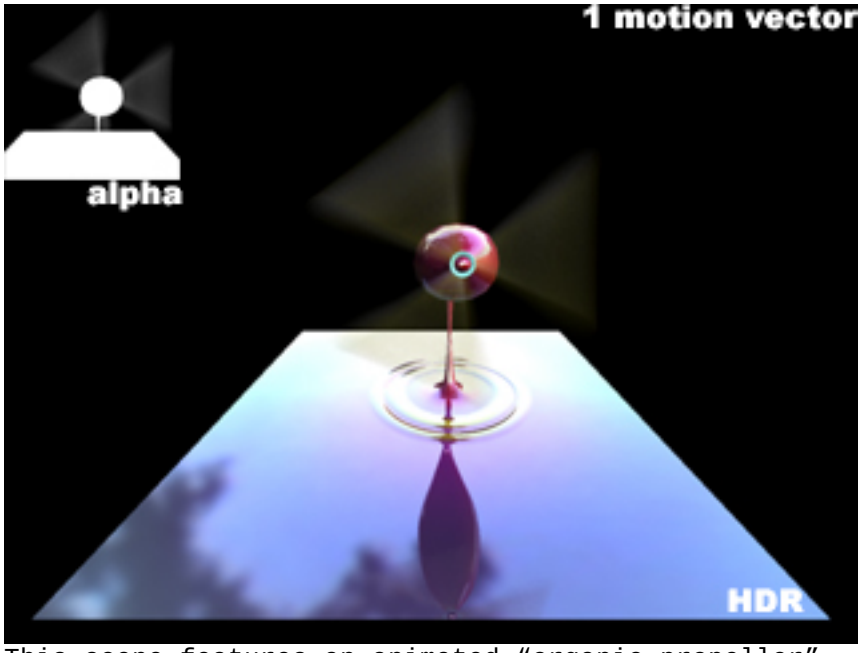
Also in mental ray you can switch between “Instance” motion blur and “Exact” motion. Instance motion blur uses a transformation matrix to define the correct directional blur. Exact motion blur uses a per vertex motion vectors to create internal motion caused by shape changes like the morphing of an object. MrfM’s based on the 3.1 implementation of the mental ray standalone. It therefore supports motion paths with up to 15 motion vectors per vertex!

To switch between these modes and to be able to set the number of motion vectors (the attribute is called “motion steps”) goto:

mental rayGlobals> your-quality-setting> motion blur> calculation

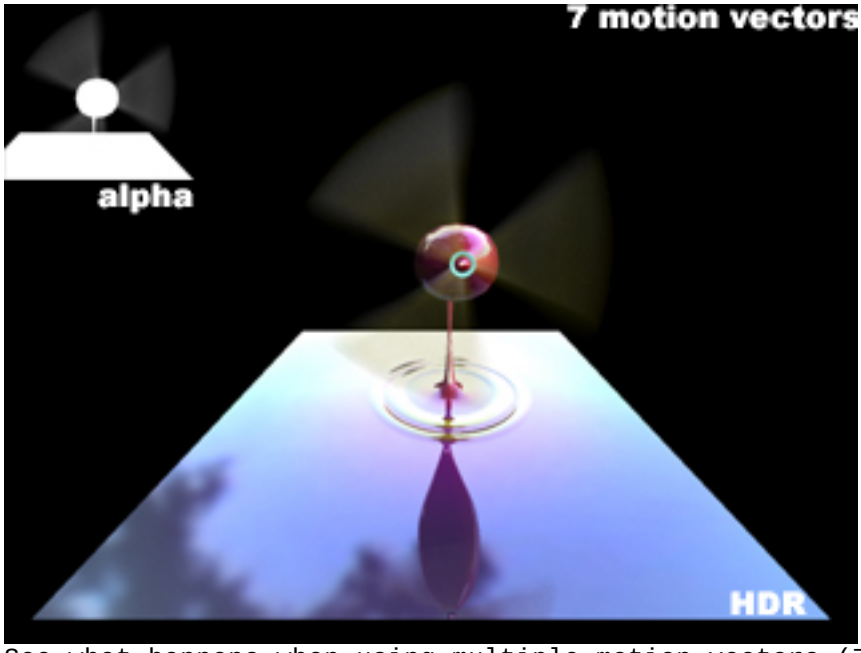
Also mental ray 3.1 allows for delaying the shutter time opening. This is useful for bidirectional motion blurring in output shaders. Parameters are available in mrfM in the same tab via the “shutter delay” attribute.

The use of multiple motion steps is useful when we have shape deformations, morphings etc. It can be very useful in all cases where you need to motion blur a propeller, an helix, an old engine of a plane rotating, and in general any rotating object. I’ve prepared an example setup scene to show this in detail.

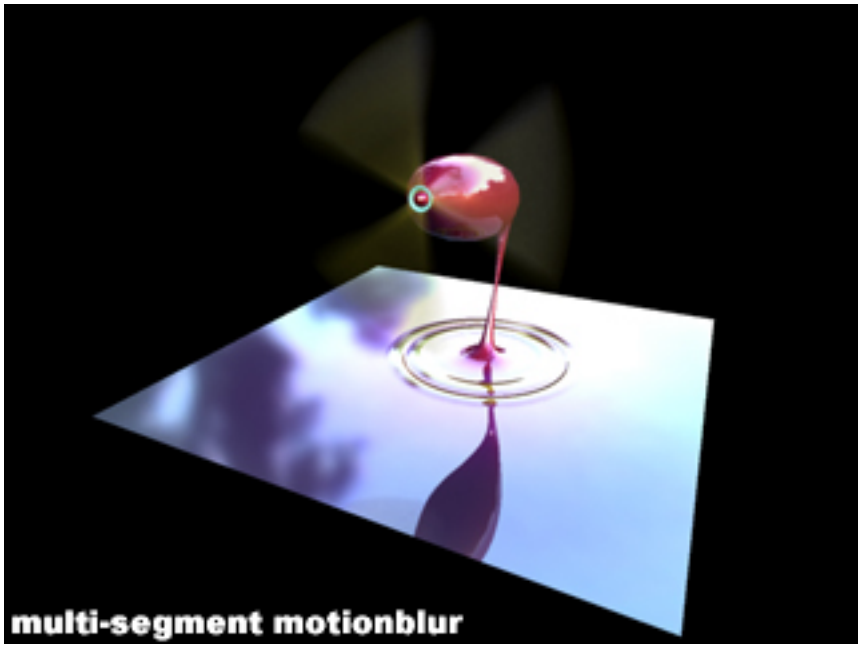


This scene features an animated “organic propeller”.
The supporting geometry is a single object obtained with a smooth proxy operation. The helix is made of three elements at 120 degrees. Also, to make this nicer an HDR environment sphere texture is mapped to the “reflected color” of the propeller. This image is rendered with 1

motion vector per pixel. The blur is not correct; (see alpha for more contrast) the rotation should produce a rounded blur an not a linear blur.



See what happens when using multiple motion vectors (7, in this case) the blur is now correct. See a different view in the next image.



At this point I would like to summarize our findings: motion blurring can be solved in many ways. I would like to represent the main concepts starting from the one we have seen until now:

- motion blur via mental rayglobals> yourqualitysetting> motion blur.

  Sampling works in the same way as the renderer sampling quality but in this case sampling is performed over time.
  Hence you have “RGBA time contrast” to control supersampling, and “motion blur by”, “shutter”, “shutter delay” to control the amount of blur and the eventual delay in time (if set to 0.5 it shifts the evaluation time to the center of the frame).

  In addition to the regular motion blur, we’ve mentioned a “fast mode” enabled if the shutter period is nonzero, and if the time contrast is set to 0 and if the min and max sampling values are set to the same value.

  Then we can choose between linear (shape translation/rotation) and exact (shape deformatiom) motion blur choosing to use up to 15 motion steps for multi segment motion blur.
  Keep in mind, if shutter is equal to 1 then motion blur length is equal to motion vectors length.
  Also, in:

  mental rayglobal> performance

  there’s an option called export shape deformation which is needed for correct motion blur of deforming objects (exact motion blur). If not using motion blur and not rendering an animation with incremental changes, switching off this option can speed up the process.

  In order not to waste computational time remember that lowering the time contrast values produces finer quality at the expense of rendertime. Increasing the global rendering sampling quality also produces better results and in this case you can keep higher time contrast values.
  You can keep the alpha (A) time contrast low if you don’t need alpha information.

- You could obtain 2d motion blur via an output custom shader which uses exported motion vector information. It is not available in the mrfM UI, but it’s achievable via shader writing. I recommend the “programming with mental ray” book from Thomas Dryemeyer if you’re interested in mental ray shading writing.

- motion blur in post is also another solution considered in production environments (and non), using 3rd part tools which are available for high end compositing packages

EXTRA FEATURES

howto activate mental ray custom shadersnodes

In mrfM 1.5 there is new support for mental ray custom shaders, which are used in all the extra features scenes below. These shaders are assembled into shading networks using Maya's Hypershade and assigned to Maya objects. Everything gets rendered in the UI renderview.

In order to ensemble these shaders in your shading networks you need to set up an environment variable:

MAYA_MRFM_SHOW_CUSTOM_SHADERS = 1

Then after (re)starting Maya you will see a collection of new nodes in the hypershade's create render node panel (or when clicking with the RMB on the window). All these nodes correspond to shaders which are explained in the online documentation:

"mental ray Shader libraries reference"

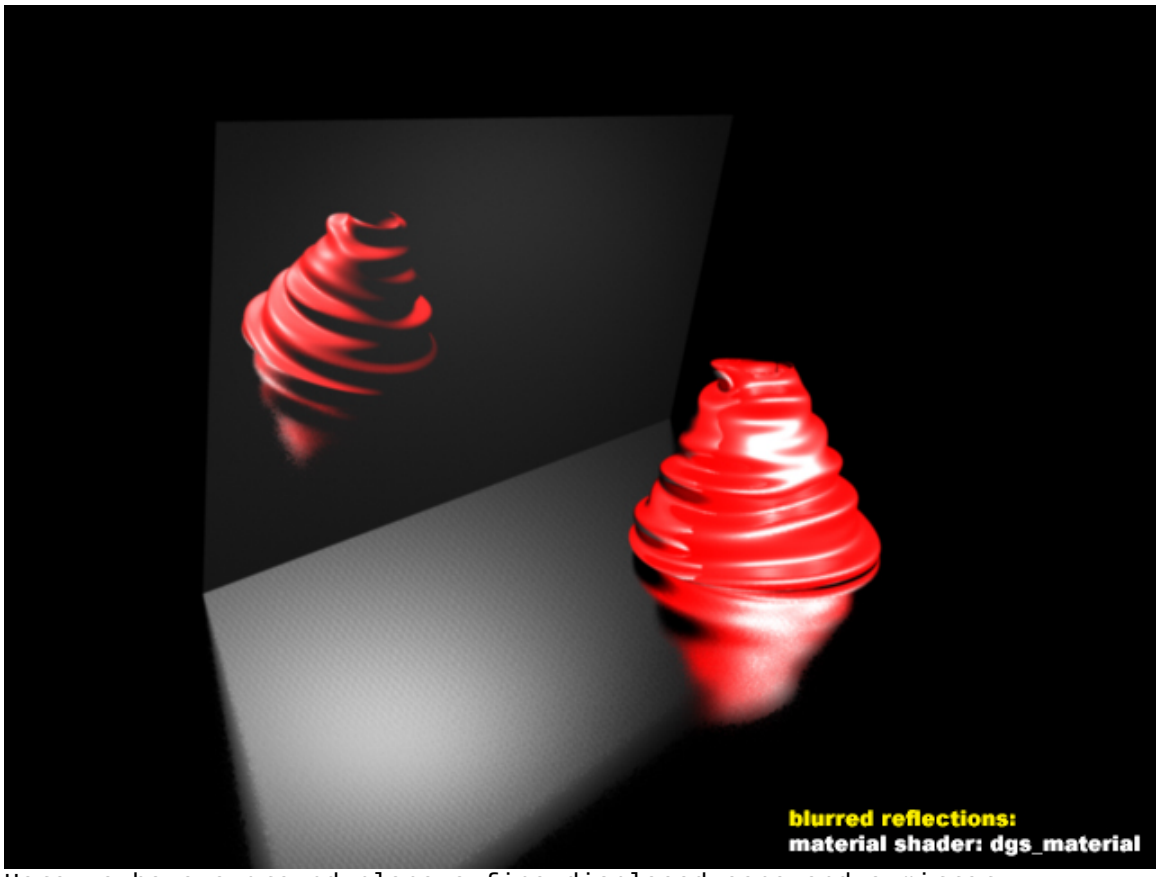
I won't explain what every shader does while I'll show how to use them, usually many of these nodes can be created and dragged and dropped to the correct "ports" available in the shading group node of every Maya material shading group (SG).
But let's see some of what we can get with these nodes in the following examples.
You'll find all the files used in this doc in their relative directory in the DVD.

Blurred reflection (Dgs_material)

Dgs_material can be useful to get the "blurred reflection" effect, a reflection that vanished/blurs with distance.
The node is available in Hypershade on:

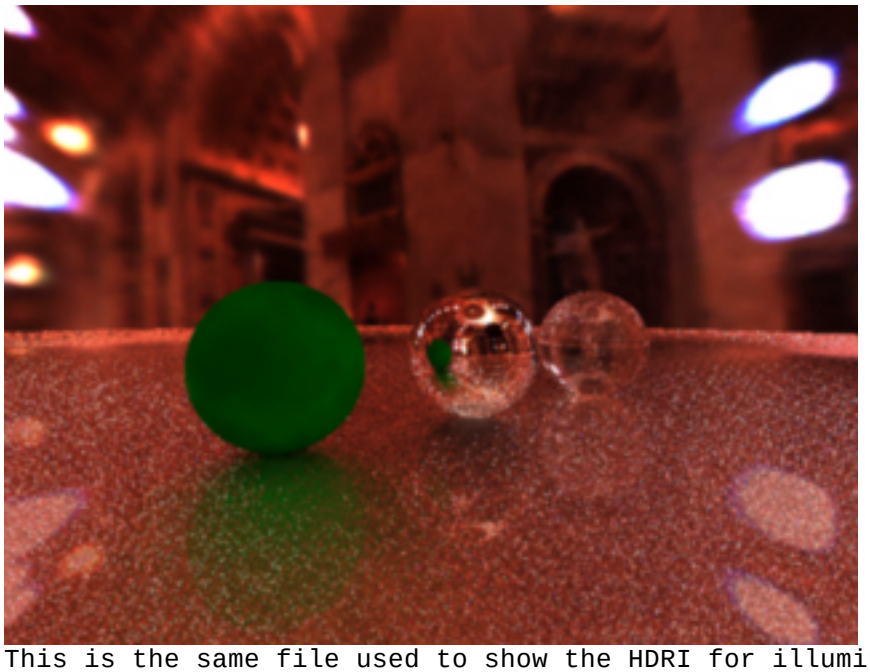
create> mental ray material> dgs_material

let's see an example:



Here we have a ground plane a fine displaced cone and a mirror.
Notice how the reflection of the fine displaced cone gets blurred on the plane and reflected correctly in the mirror.

To get the blurred reflection effect you need to assign first a classic Maya shader to the ground plane, a lambert shader for example. After this you need to select the SG node of the lambert shader and browse the mental ray tab. In it's attribute editor (everything is shown on the DVD), you'll find a "material shader" attribute where you have to map via drag'n'drop (or connection editor) your dgs_material.
You can check the parameters I've set on the file in the documentation.
Below is another simple example.



This is the same file used to show the HDRI for illumination features. Here the ground floor shader is a dgs_material, you can see again how nicely the reflection vanishes away with distance.

Contour rendering (contour_store_function_simple; contour_contrast_function_simple; contour_composite; contour_shader_simple)

Contour rendering has an entire chapter on the mental ray custom shader reference online manual and there are many shaders which can be created to obtain a whole set of contour effects; you can access these nodes via hypershade:

create> mental ray miscellaneous> contour_...

In the following example (the model geometry “doodle” is kindly from Bruno Pollet, www.br1.org) there are few steps to follow to get the contour rendering effect:

First we need to create a contrast and a store function which basically defines where and how to sample the contour (read the online docs on how contour shaders work) which we can find in hypershade:

create> mental ray miscellaneous> contour_contrast_function_simple
create> mental ray miscellaneous> contour_store_function_simple

these nodes don’t have any parameter (but you can use the non-simple functions to get additional flexibility) and they must be assigned in the relative fields via drag’n’drop or connection editor on:

mental rayGlobals> your_render_quality> contours

once this is done you need to create another node:

create> mental ray miscellaneous> contour_composite
this node must be assigned to the “output_shader” attribute in the shape node of the camera you’re using to render under mental ray tab in the camerashape attribute editor with the same system (drag’n’drop – connection editor). This node takes care of the compositing of the contour over the geometry.

Finally we need to create and assign a material shader:

create> mental ray miscellaneous> contour_shader_simple

and assign it to the “contour shader” port on the shading group (SG) of a previously assigned shader to our geometry (in this case the pinky-orange blinn). Now we are ready to render.

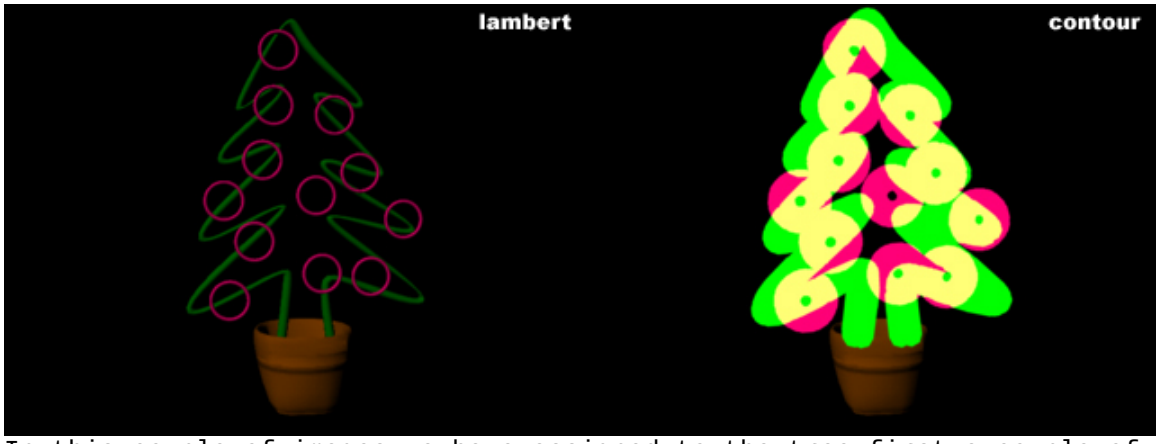
model courtesy: www.br1.org

contour rendering:
store function: simple
contrast function: simple
output: composite
shader: contour_shader_widthfromlight

In this case I’m using a shader called “width from light”: as you can see the contour works with aspects of raytracing, the ground floor is in fact a reflective background shader with shadowmasking.

faked glow (using contour_composte)
As you probably know in mrfM 1.5 Maya posteffects are not rendered with

mental ray, being shaderglow a post effect it’s actually not possible to render it’s glow in the scene. A rather simple workaround could be easily set up with contour shaders by using the contour_composite shader.
The setup is identical to the one in the previous scene. The only difference is that we are using more contour width (an attribute in the contour_shader_simple) and the “glow” flag ON in the contour_composite shader. See images below.



In this couple of images we have assigned to the tree first a couple of standard lambert shaders (on the left) then a couple of contour_shader_simple with a width of 4 (on the right), the glow flag in the contour_composite shader assigned to the camerashape node is not yet activated

et voila’, glow flag is on and now you can start rendering Broadway by night.

Volumetric fog effects (transmat, parti_volume)

The transmat material together with a parti_volume volumetric material can be used together to define a volume on which participating media are interacting with light.
In order to crete these shaders goto:

create> mental ray material> transmat
create> mental ray volumetric material> parti_volume

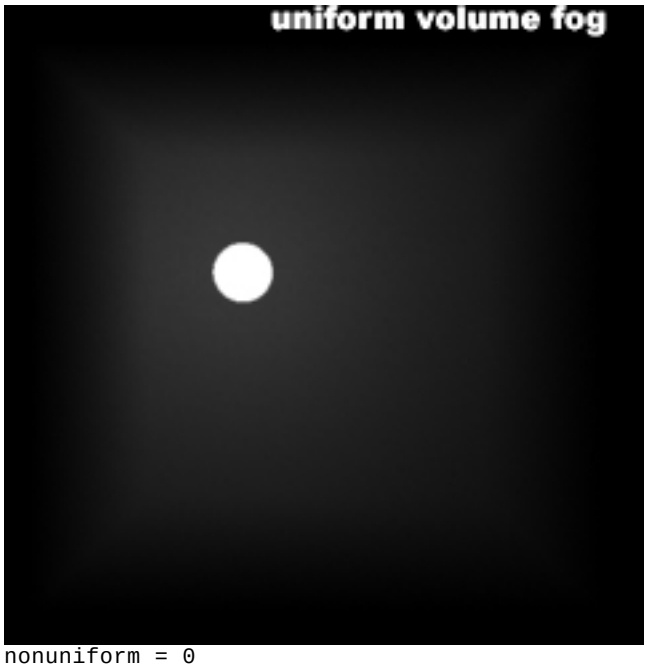
let's see an example on how to use them pointing out that you must setup a nominal connection between your light and the parti_volume shader.

In this scene we have a visible spherical real non-zero area light inside a polygon cube. To obtain the "sleepy hollow moonlight effect" first assign to the cube a Maya shader (like a lambert) then you need to assign the transmat material (which has no options) to the "material shader" port in the lambertSG shading group. The transmat is a totally transparent shader that helps define the volume where the participating media is defined (please refer to online docs and the excellent book "Rendering with mental ray" by Thomas Dryemeyer). Then the parti_volume shader must be assigned in the same manner to the "volume shader" port- always on the SG node. Here, some parameters must be changed to get a correct scattering in the volume and also you have to connect via the

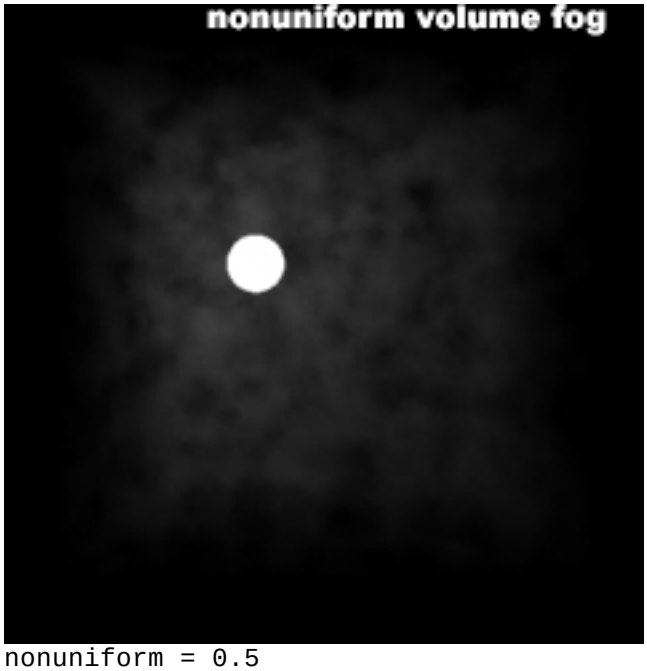
connection editor the lightShape node to the parti_volume light array attribute, it’s a nominal connection. It allows the parti_volume to know which light is interacting with the participating media. Connecting the caching attribute for example is enough, something like:

```
pointLightShape1.caching --> parti_volume1.lights[0]
```

An attribute on the parti_volume node you may want to change is “non uniform”, which defines if scattering happens uniformely or not, as in my example scene. Following some images that explain this:



nonuniform = 0



nonuniform = 0.5

Volumetric caustics (lots of shaders…)

Volumetric caustic effects can be obtained in mental ray for Maya 1.5 and rendered within the UI renderer. There is an extremely good explanation on the book “rendering with mental ray” by Thomas Dryemeyer you should read to understand well how photons work in volumes. Here I will just explain in a simple way what you need to do, please refer to the following image and the scene file on the DVD.

First, let’s see how to create the shaders we need in hypershade:

create> mental ray material> transmat
create> mental ray photonic material> transmat_photon
create> mental ray photonic material> dielectric_material_photon
create> mental ray volumetric material> parti_volume
create> mental ray miscellaneous> parti_volume_photon
create> mental ray light> physical_light

Both transmat materials have no parameters.

Ok, so now let’s define the geometry in the following image and the associated shader assignments, always performed always in the same way in the SG node. In the scene we have:

- an invisible poly cube with the same dimensions of the Cornell box to which we have connected:

  “transmat” to “material shader”
  ”parti_volume” to “volume shader”
  ”transmat photon” to “photon shader”
  ”parti_volume_photon” to “photon volume shader”

- a Cornell Box without 3 walls (top, front and behind), this just to see better the volumetric caustic effect (which detached better on a black background). To this object I have assigned standard Maya shaders.

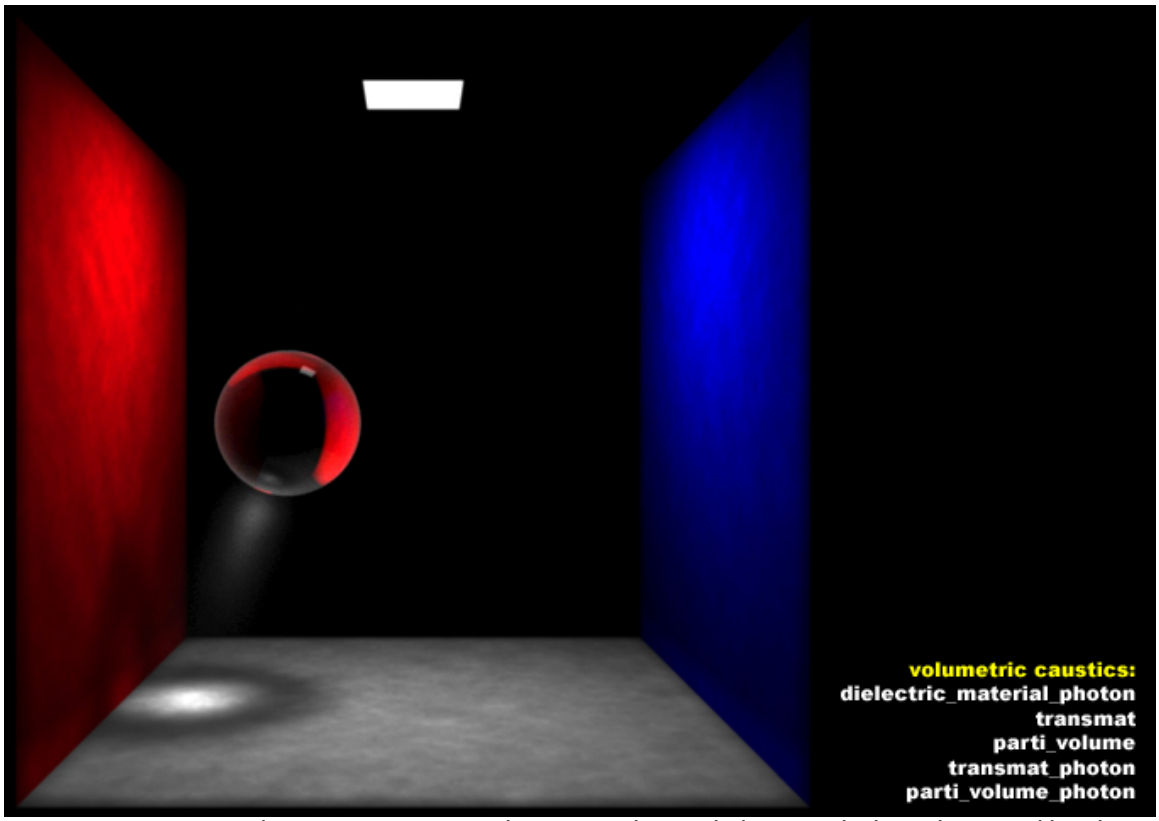
- A refractive glass sphere floating in the Cornell Box to which is assigned a glassy Maya blinn shader and:

  “Dielectric_material_photon” to “photon shader”

- a visible real non-zero arealight coming from a Maya pointlight. To this light is then assigned a mental ray light, just goto the lightShape attribute editor under mental ray> export options and connect:

  “physical_light” to “light shader”

After setting the parameters for raymarching, shading and lighting in all nodes correctly and setting up caustics as usual (check the scene for all parameters) you can finally hit render and see the progress in the renderview.



As you can see photons are now interacting with participating media in the volume defined by the transparent box. A bunch of parameters are available to control such effects and are explained in the mental ray online reference manual.

antialiasing guide

The raytracing approach to solving aliasing issues is point sampling.

There are various ways to resolve “the” problem of synthetic imagery aliasing issues.
mental ray has its own advanced antialiasing algorithm, featuring special algorithms like edge following, sample lock and jittering etc. Keep in mind that in some cases you’ll have to solve jagged edges and flickering in animation, while in some other cases you will won’t want to have accurate antialiasing for fast preview reasons.

mental ray controls sampling via the various attributes on:

renderglobal> quality setting> sampling quality:

- Contrast RGBA (after sampling with the min value, on a channel basis) if the contrast between neighboring pixels is higher than what is specified by the contrast statement, over sampling is performed until the contrast will be equal or lower. Sampling will stop when the max samples value is reached.

- Min/Max samples, the minimum and maximum number of samples to take (with lock and jittering options),
- Final spatial multipixel filtering

Lock: introduces artificial flickering to avoid static noise pattern (eg: during a slow camera pan)
Jitter: displacement of samples to avoid systematic aliasing (eg: moiré patterns)
Dithering: introduces a very small noise to avoid mach banding (eg: soft smooth faded ramp on the background)

Refer to the mental ray reference for accurate descriptions of these attributes.

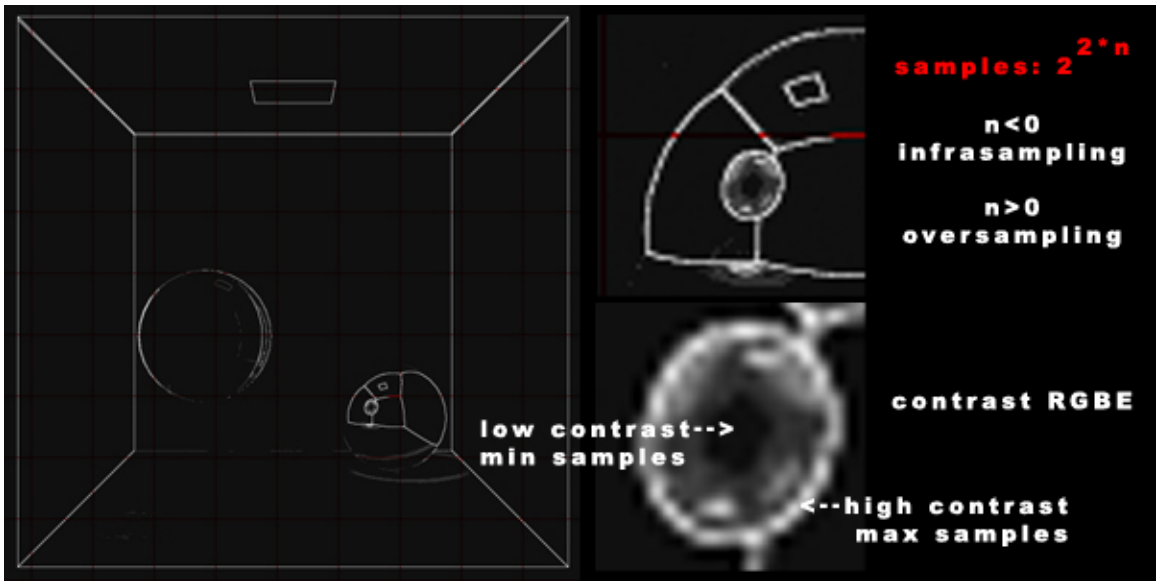
These attributes are used to control aliasing in BOTH ways, I mean, you don’t really need to have accurate anti-aliasing in all your renders, expecially in a preview phase: hence you can use these values to increase exponentially the speed of your renders!
In Maya the minimum number of sample per pixel is 1, mental ray supports a technique called “infrasampling”, basically you sample “between” pixels or at the corner of them.
Infrasampling is activated using a negative value in the “min samples” field.

IMPORTANT: infrasampling is very useful because in many cases it doesn’t matter if the initial sampling loses a long thin object because a technique called “edge following” will fix this as long as a sample is taken elsewhere.

Sampling values are specified in powers of 2 and they are bidirectional: 2^2*samples

```
samples =  1 = 2^2*1    ->  4 samples x pixel
samples =  0 = 2^2*0    ->  1 sample x pixel
samples = -1 = 2^2*-1   ->  1 sample x 4 pixels
```

To understand better see the following image, rendered with the “diagnose samples” diagnostic mode set to on:



So after the preview phase you probably want to render at serious quality. If the standard production quality provided in mental ray for Maya is not enough for your specific needs, there are 2 approaches to solve these problems:

1. oversampling, assuming you're starting from production quality, in order you can follow these steps to increase anti-aliasing:

   a) reducing the contrast RGBA values but not under a value of 0.05, then RGBA values don't need to be all the same, take in mind human eye perception is more sensitive to Green than Red and finally Blue.
   b) use a high order multipixel filtering, even if gauss 3 3 is more than enough.
   c) increasing the min max values is the last thing to do (for the exponential generation of rays) remember that generally between the min and max values there shouldn't be a difference greater than 2 units; values 1 2 or eventually 1 3 should be enough for production quality.

2. Rendering at double size

using medium sampling quality (eg contrast 0.2 0.2 0.2 0.2, minmax 2 2, Gaussian filter 2 2,). Double the size and resize of half in a compositing software which supports an advanced high order filtering algorithm (eg Mitchell). Keep in mind two other things:

   a) You can control area light lowsampling (see area light chapter).
   b) You can control cartoon rendering secondary sampling if you're using for example the contour_contrast_function_levels with the min and max levels.

Goodbye, special thanks and further reading

This documentation is finished, I hope you've enjoyed the material and thanks for arriving at this point. If you wish to contact me please do it via email:

mailto:pberto@jupiter-jazz.com

or check my website for any further information about myself:

http://www.jupiter-jazz.com

I have to thank some people for their supporting material and support:

Jos Stam @ Alias|Wavefront, for his periodic caustic textures

Ryan Meredith Jones @Alias|Wavefront, for zillions of question answered with patience

Thomas Schaedlic @ Mental Images, for extensive and detailed support

Thomas Driemeyer @ Mental Images, for his excellent book "rendering with mental ray"

Bruno Pollet @ www.br1.org for the model "doodle" and the animation on the DVD "clothdog and the mysterious ball"

Paul Debevec @ www.debevec.org for its HDR textures

and finally some bibliography and further reading:

- Thomas. Driemeyer - Rendering with mental ray
- Thomas Driemeyer - Programming with mental ray
- mental ray 3.1 reference manual
- mental ray for Maya 1.5 documentation
- Kelvin Sung, James Craighead, Changyaw Wang, Sanjay Bakshi, Andrew Pearce, and Andrew Woo - Design and implementation of the Maya renderer
- Andrew Woo – Aliasing artifacts in Maya: a technical overview
- Andrew Pearce, Kelvin Sung - Maya Software Rendering: a technical overview
- Maya rendering courseware
- Maya rendering online documentation