mental ray – irradiance particles

Document version 1.0
December 4, 2007

Introduction

Irradiance Particles is the name of a new method to calculate global illumination in mental ray, with significant advantages compared to other algorithms.
The most important are:

- Easy set up
- Fast, compared to other algorithms

This document gives a short Introduction to Irradiance Particles.
You will get an idea of the basic concept and learn how to setup and tweak a simple scene with Irradiance Particles in mental ray.

Basics

Computing global illumination with irradiance particles is split in different passes.
Before rendering starts, importons are shot from the camera into the scene.
These importons are collected as a new kind of particles, called "irradiance particles", which actually hold information about the direct illumination coming at their position and possibly the indirect one as well, hence the name "irradiance".
The particles are organized in a structure called "particle map".
Depending on how many light bounces we choose to include, several other passes are processed to also include the indirect irradiance to these particles.
If interpolation is used, a final preprocessing step is done in which irradiance is collected for every irradiance particle in order to be used later during rendering.
After this preprocessing is done, the particles are possibly saved in a file.
During rendering, this particle map is used to estimate the indirect illumination (the irradiance) for every shading point.

Set up the scene

Render the Scene, shipped with this tutorial.

**The room is quite dark,
it is only lit by the sun.
(direct illumination only)**

relative rendering time: 1

**Insert the line:
"irradiance particles" on
into the options block and render again.**

**This line activates the "irradiance particles"
with default settings.
The room is far more lit, because there is one
indirect illumination pass included.**

rrt: 5

Of course there are other string options related to "irradiance particles".
The most important are:

```
"irradiance particles indirect passes"    int         default 0
"irradiance particles rays"               int         default 256
"irradiance particles scale"              scalar      default 1.0

"irradiance particles interpolate"        string      default "always"
"irradiance particles interppoints"       int         default 64

"irradiance particles env"                bool        default  on
"irradiance particles env scale"          scalar      default  1.0
"irradiance particles env rays"           int         dynamic default

"irradiance particles file"               string      default  ""
"irradiance particles rebuild"            bool        default  on
```

Most of them are quite easy to understand, but to get a feeling, it is always a good idea to play with them.

Rays

The `"irradiance particles rays"` option determines how many rays are shot from a point to gather the illumination information.
The contribution of these rays is then averaged, which means:
The more rays are shot the more accurate is the light situation.
First, without interpolation:

“irradiance particles rays” 2
"irradiance particles interpolate" “never"

“irradiance particles rays” 256
“irradiance particles interpolate” “never”

rrt: 29

“irradiance particles rays” 512
“irradiance particles interpolate” “never”

rrt: 45

The First thing that is obviously is that rendering times are very long.
This is because interpolation is switched off, which in fact forces mental ray to estimate the irradiance by shooting rays at every shading point (the number of rays to shoot is given by the `"irradiance particles ray"` string option).
This is the most precise way to calculate indirect illumination with irradiance particles, but also the slowest.

The first image is very noisy.
At every shading point, mental ray looks in just 2 directions for irradiance particles, which is far too few.
For the third image mental rays samples the particle map with 512 rays at every shading point, this is why it took 45 times longer than the original image (the one rendered without irradiance particles).
We can greatly reduce this time, by using interpolation.

Interpolation

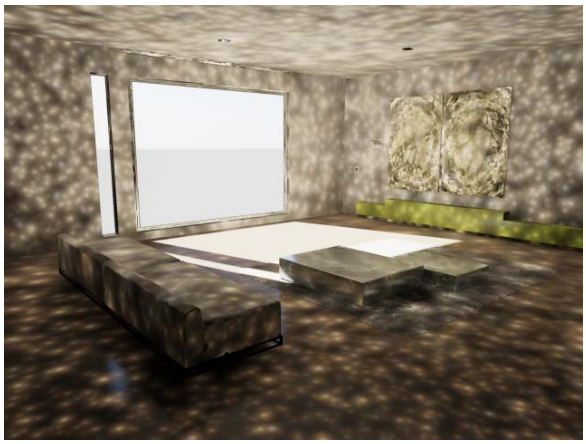
There are two string options related to interpolation:

```
"irradiance particles interpolate"  "always" / "never"
"irradiance particles interppoints"  int
```

Interpolation is enabled by setting `"irradiance particles interpolate"` to `"always"` and disabled by setting it to `"never"`, as it was done in the chapter “Rays”.

With interpolation mental ray adds an extra final irradiance pass in the preprocessing step in which it samples the particle map to gather the illumination information for each one of the irradiance particles, which will be later used as interpolation points.
When mental ray has to compute the irradiance at a shading point during rendering, it looks for the n closest irradiance particles, where n is the number, given in the `"irradiance particles interppoints"` string option and computes the weighted sum of those irradiances.
This is clearly much faster than tracing the rays for every shading point.

If the same scenes are rendered but this time with interpolation on by setting `"irradiance particles interpolate"` option to `"always"`,
The results differ, also in rendering times.



“irradiance particles rays” 2
“irradiance particles interpolate” “always”

rrt: 7

“irradiance particles rays” 256
“irradiance particles interpolate” “always”

rrt: 8

“irradiance particles rays” 512
“irradiance particles interpolate” “always”

rrt: 11

Factor 8 for the second image is nice, we take it.

Now a good number of interpolation points should be found.
Until now always 64 interpolation points, the default, are used,
but what happens if this parameter is varied.

“irradiance particles interppoints” 3
“irradiance particles rays” 256
“irradiance particles interpolate” “always”

rrt: 6

“irradiance particles interppoints” 64
“irradiance particles rays” 256
“irradiance particles interpolate” “always”

rrt: 8

“irradiance particles interppoints” 256
“irradiance particles rays” 256
“irradiance particles interpolate” “always”

rrt: 12

As one can see 3 points (the minimum) are far to less, the image get artifacts resulting from the fact that the irradiance at a shading point is computed by building the weighted sum of just these three points.

These artifacts are gone if we use 64 interpolation points because every single interpolation point contributes less irradiance to the result.
Increasing the number of points only results in longer rendering times.

The trick is, to use enough Interpolation points, to not get artifacts, but not using too many, otherwise a lot of detail gets lost (and it takes longer).
The default 64 works quite well for most scenes.

Environment

Irradiance Particles are also able to gather light information from environment shaders.
There are 3 parameters related to the environment.

```
"irradiance particles env"          bool        default on
"irradiance particles env rays"     int         dynamic default
"irradiance particles env scale"    scalar      default 1.0
```

The first option enables/disables the environment for irradiance particles, the second is the number of rays to be used to evaluate the irradiance coming from the environment.
The dynamic default for `"irradiance particles env rays"` means that it is by default the same number that is set with the `"irradiance particles rays"` string option.
It is there for fine tuning reasons.
This default works well for outdoor scenes, but for indoor scenes, where the environment is only seen through a window, it is better to just increase the `"irradiance particles env rays"` and leave the `"irradiance particles rays"` value as it is, or set `"irradiance particles env" off` and use a portal light instead.