mental ray – irradiance particles

Document version 1.1
March 20, 2008

Introduction

Irradiance particles is the name of a new method to calculate global illumination in mental ray®, which is easy to set up and faster than other algorithms.

This document provides a short introduction to irradiance particles that describes the underlying concept and shows how to setup and tweak a simple scene with irradiance particles in mental ray.

Basics

Computing global illumination with irradiance particles is performed in multiple preprocessing steps.

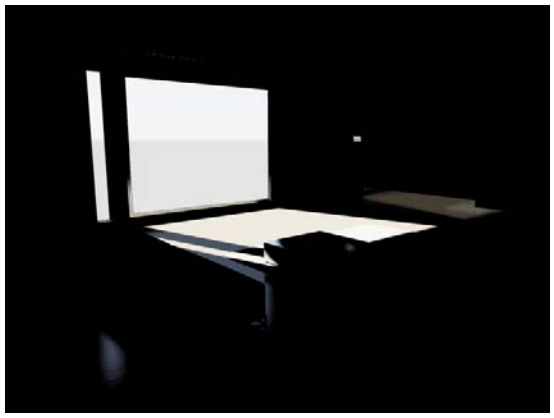
1) Importons are shot from the camera into the scene.
2) Importons are collected as irradiance particles. They are organized in a structure called "particle map" and hold information about direct illumination at their position.
3) Depending on how many light bounces you choose to include, several other passes are processed to also include the indirect irradiance of these particles.
4) 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 may be saved in a file.

During rendering, the particle map is used to estimate the indirect illumination (the irradiance) for every shading point.

Render the scene

In this tutorial we render a simple indoor scene with irradiance particles. To compare rendering times, we first set the irradiance particles option *off* (figure1).

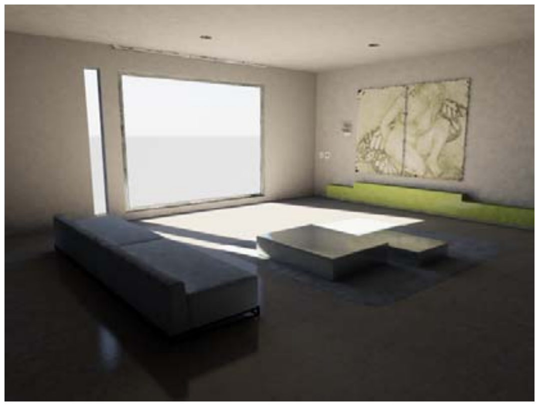


relative rendering time: 1
Figure 1

The room is quite dark, it is lit by the sun`s direct illumination only.

To activate irradiance particles using default settings, we need to add the following lines to the options block:

```
"irradiance particles"  on
"importon"              on
```



rrt: 2.3
Figure 2

The room in Figure 2 is far better lit because one indirect illumination pass is included.

Irradiance particles options

Of course there are other 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 it is recommended to try them and experiment with them.
This is what we do in this tutorial.

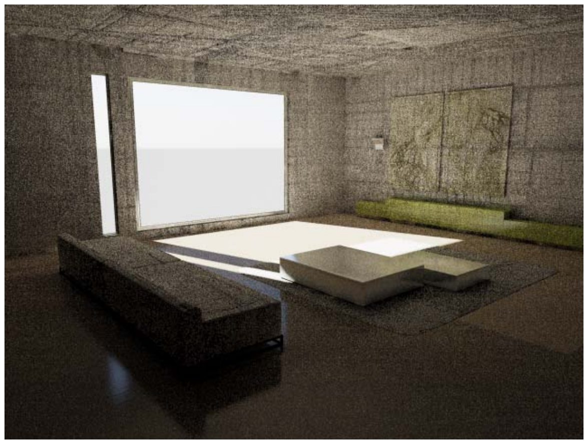
Importons

In the context of irradiance particles, you can think of importons as locations where indirect light can be stored.
Therefore it is important to cover *all* parts of the scene seen by the camera with importons including those *indirectly* seen, for example, through a mirror.
For the second case we need to allow them to bounce in the scene.
We do this by setting `"importon trace depth"` to a nonnull value. Most often higher values are useful, but the right trace depth depends on the scene.

To avoid proliferation of irradiance particles the `"importon traverse"` option is always set to `"off"`.This value cannot be set.

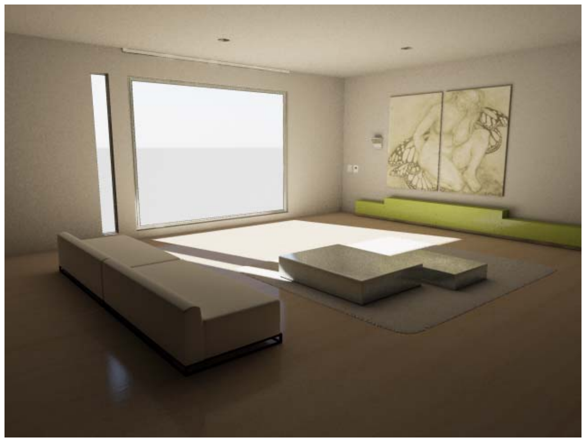
Rays

The `"irradiance particles rays"` option determines how many rays are shot from a point to gather illumination information.
The contribution of these rays is averaged. This means that shooting more rays produces more accurate lighting.
Figures 3–5 show different results from varying the number of rays without using interpolation (by setting `"irradiance particles interpolate"` to `"never"`).



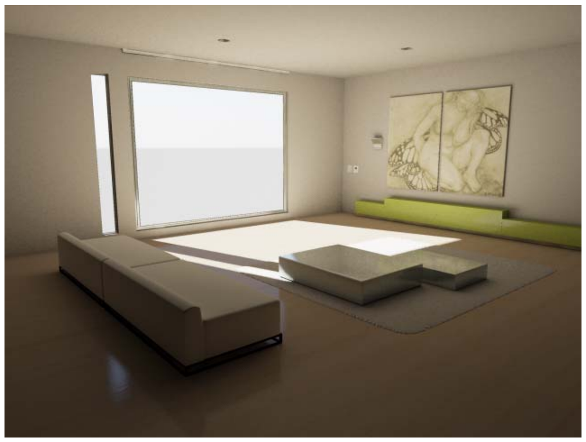
“irradiance particles rays” 2
"irradiance particles interpolate" “never"

rrt: 2.6
Figure 3



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

rrt: 25.3
Figure 4



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

rrt: 50
Figure 5

Rendering times for figures 3, 4 and 5 are very long because interpolation is switched off, which forces mental ray to estimate the irradiance by shooting rays at every shading point (the number of rays to shoot is determined by the `"irradiance particles ray"` option).
This is the most precise way to calculate indirect illumination with irradiance particles, but is also the slowest.

The image in figure 3 is very noisy.
At every shading point, mental ray sampled two directions for irradiance particles, which is far too few.
To produce the image in figure 5, mental ray sampled the particle map with 512 rays at every shading point, this is why it took 50 times longer than the original image to render.
Rendering time can be significantly shortened by using interpolation.

Interpolation

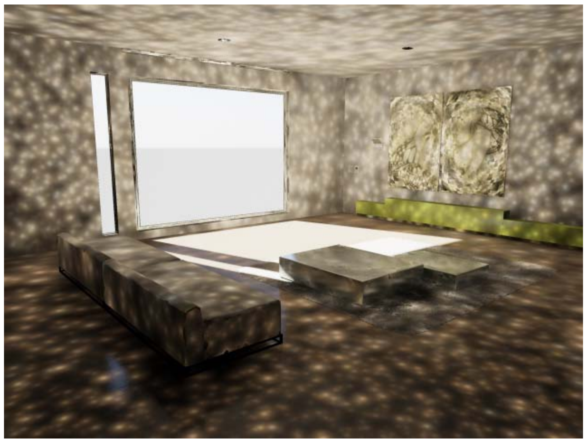
There are two string options related to interpolation:

```
"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 section "Rays".

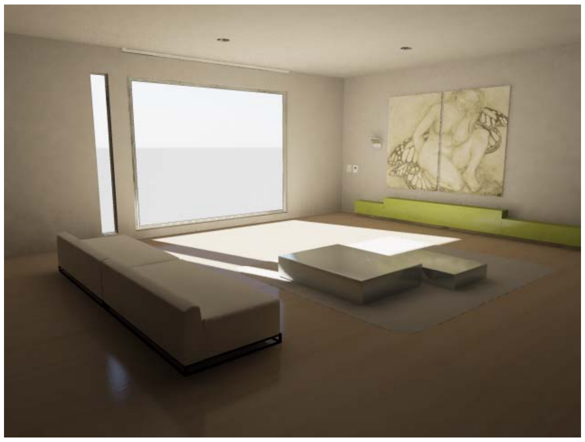
By using interpolation, mental ray adds an extra final irradiance pass in the preprocessing step where it samples the particle map to gather the illumination information for each irradiance particle, which will later be used as interpolation points.
When mental ray computes the irradiance at a shading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

Rendering the same scenes, but this time with interpolation, (by setting `"irradiance particles interpolate"` option to `"always")` results in much shorter rendering times.



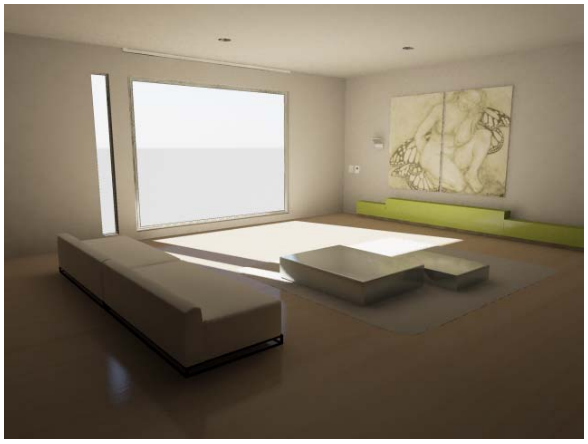
“irradiance particles rays” 2
“irradiance particles interpolate” “always”

rrt: 1.6
Figure 6



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

rrt: 2.4
Figure 7

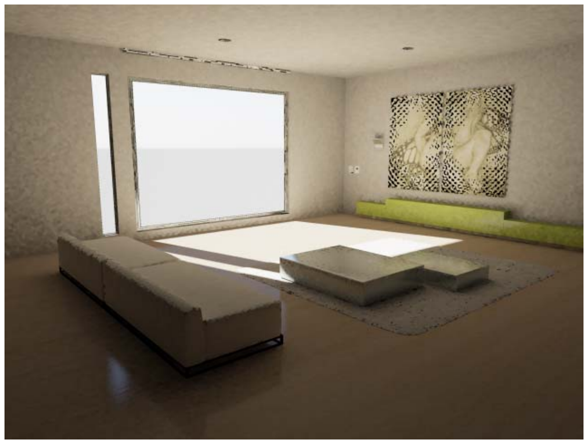


“irradiance particles rays” 512
“irradiance particles interpolate” “always”

rrt: 3.2
Figure 8

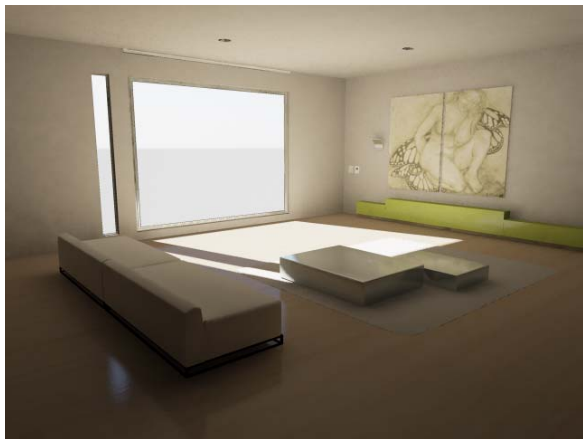
Factor 8 for the image in figure 7 is nice, we take it.

The next step is to determine an optimal number of interpolation points.
In the preceding examples, 64 interpolation points, the default, were used.
The following figures show what happens when this parameter is changed.



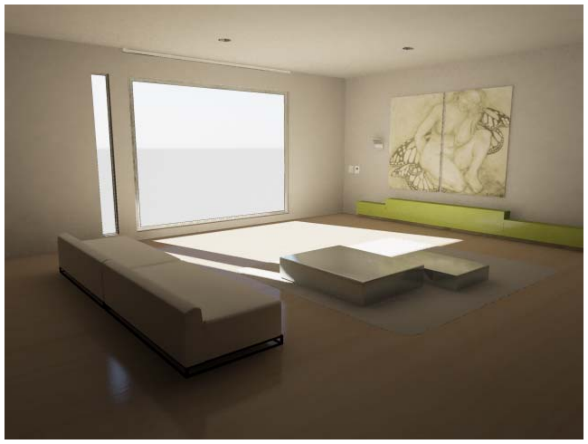
“irradiance particles interppoints” 3
“irradiance particles rays” 256
“irradiance particles interpolate” “always”

rrt: 2.4
Figure 9



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

rrt: 2.4
Figure 10



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

rrt: 4
Figure 11

Three interpolation points (the minimum) are too few, the image gets artifacts because the irradiance at a shading point is computed by building the weighted sum of these three points only.

When 64 interpolation points are used, the artifacts disappear because every single interpolation point contributes less irradiance to the result.
However, increasing the number of points results in longer rendering times.

Use enough Interpolation points, to avoid artifacts and not lose important detail. The default value, 64, works quite well for most scenes.

Additional Options

Environment

Irradiance particles are also able to gather light information from environment shaders.
Three options are 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
```

`"irradiance particles env"` enables/disables the ability to gather light from the environment shader.
`"irradiance particles env scale"` is simply a scale factor and
`"irradiance particles env rays"` is the number of rays used to evaluate the irradiance coming from the environment shader.
For fine tuning, the dynamic default for `"irradiance particles env rays"` is the same number that is set for the `"irradiance particles rays"` option.
This default works well for outdoor scenes, but for indoor scenes, where the environment is only seen through a small window, it is better to increase the `"irradiance particles env rays"` value and leave the `"irradiance particles rays"` value unclamped.
When possible, best solution is to switch `"irradiance particles env"` off and use a “mia_portal_light” instead.

IP Map

Sometimes you do not want to rebuild the entire particle map.

For this case, it is also possible to write the particle map to disk for reuse later on. The two options are consistent with photon or finalgather maps.

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

If the file specified in `"irradiance particles file"` exists and `"irradiance particles rebuild"` is `"off"`, mental ray reads the file instead of computing a new particle map. If mental ray cannot find the file, it computes the map and saves it under the given name.
By setting `"irradiance particles rebuild"` to `"on"` mental ray always computes the map.