

Realistic Light Modelling with Photon Mapping in **POV-Ray**.

Josh McCoy

July 25, 2006

Introduction

Photon Mapping

In computer graphics, photon mapping is a global illumination algorithm based on ray tracing used to realistically simulate the interaction of light with different objects. Specifically, it is capable of simulating the refraction of light through a transparent substance, such as glass or water, diffuse interreflections between illuminated objects, and some of the effects caused by particulate matter such as smoke or water vapor. It was developed by Henrik Wann Jensen.

In the context of the refraction of light through a transparent medium, the desired effects are called caustics. A caustic is a pattern of light that is focused on a surface after having had the original path of light rays bent by an intermediate surface. An example is a glass of wine on a table. As light rays pass through the glass and the liquid, they are refracted and focused on the table the glass is standing on. The wine in the glass also produces interesting effects, changing the pattern of light as well as its color.

With photon mapping, light packets (photons) are sent out into the scene from the light source. Whenever a photon intersects with a surface, the intersection point, incoming direction, and energy of the photon are stored in a cache called the photon map. As each photon is bounced or refracted by intermediate surfaces, the energy gets absorbed until no more is left. We can then stop tracing the path of the photon. Often we stop tracing the path after a predefined number of bounces in order to save time. One of the great advantages of photon mapping is the independence of the scene's description. That is, the scene can be modeled using any type of geometric primitive as for instance triangles, spheres, etc. [wikipedia]

POV-Ray

POV-Ray, or Persistence of Vision Raytracer, is a rendering and ray tracing application with an open license and freely available source code. Specializing in high quality 3D rendering, POV-Ray makes use of many computationally expensive algorithms when rendering a scene.[povray.org]

Preparing Environment for Running POV-Ray

LAM/MPI Environment

To run POV-Ray in parallel, a working LAM/MPI environment is needed. BCCD provides a user, lambccd, with such an environment. SSH to a machine running the BCCD as the lambccd user (the password should be "bccd"):

```
ssh lf0 -l lambccd
```

Passwordless Authentication via SSH

Logon to each machine with which you want to run POV-Ray as the lambccd user chose "yes" when prompted about starting a heartbeat process and run `bccd-allowall`:

```
bccd-allowall
```

On the machine from which you want to start the jobs, gather a machine list:

```
bccd-snarfhosts
```

Confirm your machine list:

```
bccd-checkem
```

Obtaining the Curriculum Module

Use the `list packages` command to bring up a list of available curriculum modules and software:

```
list-packages
```

Select the `Photon Mapping` package and press enter. The curriculum module should now be installed in the home directory of the lambccd user. Enter the `povray` directory:

```
cd povray
```

Logging Waltime

As you progress, record the time it takes to render the three scenes using each rendering method in the following table:

	Ray Tracing	Photon Mapping	Dispersion
Glass Sphere			
Diamond Box			
Water Torus			

Modeling a Glass Sphere

The default rendering method of POV-Ray is ray tracing. Run the `sphere-rt.sh` script

```
./sphere-rt.sh
```

After the rendering is complete, view the animated gif, `sphere-rt.gif`, via `firefox` (in `firefox`'s File menu, select open file and set the filter to image files). To start `firefox`, use the following command:

```
firefox &
```

Notice how ray tracing reproduces a translucent sphere while refracting the light as it passes through the glass.

If the glass sphere was seen in a similar scene with real world light interactions, you would see light light being focused by its refraction through the glass. As mentioned in the introduction, this focusing of light is called a caustic. One benefit of photon mapping is that it can form these caustics. Now, render the same glass sphere with photon mapping enabled by running the `sphere-pm.sh` script:

```
./sphere-pm.sh
```

View the results in `firefox` and notice the caustics produced by the sphere.

Dispersion, or the separation of the wavelengths of light as the light is refracted through media, is shown in the next sphere to be rendered. Run the `sphere-dis.sh` script:

```
./sphere-dis.sh
```

View the animated gif and examine the caustics and notice the dispersion (thin lines of rainbow-like color) at their edges.

Modeling a Diamond Box

Run the script to render a diamond box with ray tracing:

```
./box-rt.sh
```

Notice the complexity of the light refracted in the box itself.

Finding points of collision of rays and photons with rendered objects (a process called collision detection) is a crucial computation to rendering in POV-Ray. As was the case with the set of sphere renderings, detecting collisions with a sphere is a simple algorithm and takes little time to compute. Detecting collisions with a simple geometric object, such as a box, takes slightly more computation. Run the same box with photon mapping enabled:

```
./box-pm.sh
```

The time it took to render this scene is much longer in comparison to the time used to render the glass sphere with photon mapping. This is due to the amount of collision detection. Each time a photon hits a face of the box, one of two additional collision detections can take place: one if the photon is reflected by the surface and another if the photon is refracted. Each photon or ray is traced through a scene while it encounters reflective or translucent surfaces until it finds no collision is detected (it shoots off into empty space) or the medium it hits does not reflect or call for further tracing.

When a medium disperses light during refraction, additional photons corresponding to the different wavelengths of the dispersed light need to be modeled. Run the diamond box with dispersion enabled to see the increased workload needed to render dispersed light:

```
./box-dis.sh
```

Modeling a Torus Made of Water

Render the water torus with ray tracing:

```
./torus-rt.sh
```

Without photon mapping, the shadow of a non-white and translucent object takes the color of that object. A more complex and believable shadow can be rendered via caustics. This is shown in the output of the rendering of our second torus:

```
./torus-pm.sh
```

The resultant animation looks much more realistic than the ray traced version. Also noticeable is the sharp increase in rendering time. The torus is a more complex object than either the box or the sphere and has many more faces for the photons to interact with. Now render the same scene with dispersion enabled:

```
./torus-dis.sh
```

The results should look identical to the previous animation. This is due to the full spectrum, white light being filtered through a color medium. Only a single wavelength of light, corresponding to the wavelength of the color of the water, transmits through the torus. Since there is only a single wavelength of light being refracted by the torus, there are no separate colors of light to disperse. By looking at the time it took to render, it is evident that POV-Ray does not perform dispersion calculations on light of a single wavelength.