Global Illumination: Solving the Rendering Equation by Photon Mapping

rrti

May 18, 2014

Abstract

This article accompanies a basic Jensen-style photon mapper, built as part of an elective course project to investigate global illumination methods. We give a quick review of the theoretical background behind photon mapping, and sketch the workings of our mapper implementation. Several shortcomings inherent to this method are also listed.

1 Introduction

Realistic image synthesis has been a major field of science ever since the first computers were able to display graphical output, with applications in data visualisation, movies, games, etcetera. Nowadays we have reached a point where it is possible to generate high quality life-like images within a reasonable amount of time even on consumer-level computers. This enables us to perform more complex computations that simulate light interaction with underlying models derived from optics and physics, incorporating various light-transport effects observed in the real world, such as soft shadows, indirect illumination and caustics. These effects can be achieved by solving Kajiya's rendering equation[1], an integral in which the radiance [1] leaving a point on a surface is given as the sum of emitted plus reflected radiance. Evaluating this integral is the task of all global illumination[2] algorithms.

In this paper, we will outline a technique to solve the rendering equation called photon mapping. Section 2 first introduces the classic ray tracing algorithm on which photon mapping is based, as well as the reflection models that simulate light-surface interaction, and a convenient notation for light-transport paths. In section 3, the components of the photon mapping method are described. In section 4, our approach to implementing this algorithm are discussed, as well as several of its underlying assumptions. Next, section 5 provides an analysis of the implementation with respect to its ability to approximate the rendering equation. Results are shown in section 6 and we end with a conclusion where we discuss the pros and cons of photon mapping and some directions for optimization.

[1] Radiance is a measure for the amount of light passing through or emitted from an area on a surface; specifically, the radiant flux per (unit) solid angle per (unit) projected area. We will not treat radiometric concepts in detail.

[2] An umbrella term for algorithms that evaluate both direct and indirect (ie. the illumination of surfaces by light reflected from other surfaces[4]) lighting.

2 Background

The first attempts at realistic image synthesis were made with ray casting[3] by Appel as early as 1968 [2]. Unlike in later methods, cast rays are not traced further into the scene upon intersecting an object. This precludes the possibility of rendering indirect illumination effects such as reflections. However, ray casting forms the basis of a whole class of image-order algorithms that compute per-pixel radiance by following the path of light along rays. We present a brief overview of recursive ray tracing, arguably the most elegant global illumination method descended from ray casting.

Introduced in 1980 by Whitted [3], ray tracing extends ray casting by allowing the primary rays shot through the image plane into the scene from the camera to recursively spawn new rays when intersecting a surface. Typically, the surface BRDF[4] is assumed to be some linear combination of Lambertian-diffuse and specular components, so that computationally simple models (eg. a reflection about the surface normal in the specular case) for determinining the direction of a new ray suffice. The depth of the recursion tree is bounded by some constant k to guarantee termination. Note that the diffuse part of a material does *not* act as a source for secondary rays, since an accurate evaluation of a Lambertian (or other near-isotropic) reflection distribution function would be far too expensive. Consequently, ray tracing is limited to specular reflections or transmissions[5], although stochastic diffuse reflections can be integrated by means of Monte Carlo sampling. Finally, geometrically correct hard shadows are computed by a binary visibility function V that determines line-of-sight to a point on a surface with respect to a light source. "Soft" shadows with a penumbra region can be realized by modelling area- rather than point-lights and again applying Monte Carlo sampling (of V).

To facilitate comparing the expressive power of various global illumination algorithms in terms of the light-paths they can capture (and hence, the quality of their solutions to the rendering equation), Heckbert [4] devised a simple notation that borrows from regular expressions. In this notation there are four types of light-path vertices: L, S, D, and E. These respectively denote a light-source, specular surface interaction, diffuse surface interaction, and the eye (camera), and may be combined via the usual operators $+$, $*$, ?, and $|$. A standard raytracer without any Monte Carlo aspects is then summarized by the expression `LD?S*E`, which directly shows that eg. a sequence of D events[6] can not be simulated.

The rendering equation itself, whose physical basis lies in the laws of conservation of energy, has the following appearance:

$$L_o(x, \overrightarrow{w}) = L_e(x, \overrightarrow{w}) + L_r(x, \overrightarrow{w})$$

[3] In computer graphics, the term ray casting has several different (though related) meanings. The most general usage refers to the problem of determining the first object intersected by a ray.

[4] The bi-directional reflectance distribution function, which maps incoming irradiance to outgoing radiance for every pair of hemispherical directions at any surface point.

[5] Also named "refractions", which are rays generated according to Snell's law for refractive materials.

[6] ie. diffuse interreflections, which are responsible for the "color bleeding" phenomenon.

with

$$L_r(x, \overrightarrow{w}) = \int_\Omega f_r(x, \overrightarrow{w}', \overrightarrow{w}) L_i(x, \overrightarrow{w}')(\overrightarrow{w}' \cdot \overrightarrow{n}) d\overrightarrow{w}'$$

Here L_o is the outgoing radiance at position x in direction $\overrightarrow{w}$, expressed as the sum of the self-emitted radiance L_e and the (cosine-weighted) proportion of reflected incoming radiance L_r integrated over all incident hemisphere directions $\Omega\overrightarrow{w}'$ above x. The BRDF mentioned previously is given by f_r, which has the canonical form $\frac{dL_o(x,\overrightarrow{w_o})}{L_i(x,\overrightarrow{w_i})\overrightarrow{w_i}\cdot\overrightarrow{n}\, d\overrightarrow{w_i}}$. As formulated above the equation is incomplete, since several illumination effects (eg. fluorescence and polarization) are parameterized both by the wavelength λ of light and the passage of time t. Nevertheless, most phenomena of interest can be rendered despite these independence assumptions. Also note the equation's implicit recursive structure: for any surface, L_i depends on L_o from a possibly infinite number of *other* surfaces, and therefore analytical solutions do not exist.

In our implementation we use Phong reflection as a model for local illumination. Unlike the physical approach of radiance estimation in global illumination, this model is based on an empirical observation made by Bui Tuong Phong [6] that shiny surfaces have specular highlights based on the observers orientation and the position of the light source(s). Our reason to include this model for reflection is that its fairly cheap to compute, unlike the more physically correct reflection models.

3 Photon Mapping

The method of photon mapping was developed by Henrik Wann Jensen and serves as an extension of the standard ray-tracing to solve the rendering equation and improving image quality and computation time compared to other global illumination techniques like path-tracing [5]. The main idea is to store photons in a spatial datastructure after being emitted from some light source and interaction with the scene. By storing them separately in a datastructure, the illumination information is decoupled from the geometry unlike most other global illumination algorithms. After building up the map, the stored photons are used to estimate radiance for the rendering step.

3.1 Photon Tracing

To build up the photon map, we need to have light sources emitting photons. This emission is dependent on the shape of the light source, and within our implementation we restricted the shapes to diffuse point lights and spherical area lights both using the same method for emission. Because we are limited to a number of photons for emission, a Monte Carlo technique called rejection sampling is used to simulate the uniform emission (provided that the randomizers used yield uniformly distributed values). The idea is simple: sample random points until a certain condition is met. In the case of our light sources, this condition is that we sample points within an unit cube; if it is the case that these points in the unit cube are also inside the unit sphere, we use the point as a direction for our photon emission.

For the interaction between photons and the objects in the scene we use the Monte Carlo technique Russian Roulette. This can be seen as an importance-sampling technique because of the probabilistic model it uses. Given a surface, we define properties for diffuse, specular and refractive characteristics for interaction (absorbance is defined implicitly in these properties). In the case of 3 color bands as with our implementation, the average reflectance needs to be used to define the intervals for Russian Roulette:

$$\rho_{d,avg} = \frac{\rho_{d,r} + \rho_{d,g} + \rho_{d,b}}{3}$$

$$\rho_{s,avg} = \frac{\rho_{s,r} + \rho_{s,g} + \rho_{s,b}}{3}$$

Ensuring that each of these averages are in the range $[0, 1]$ and sum to 1 or less, Russian Roulette decides whether a photon will be reflected, transmitted or absorbed with the aid of a randomizer ψ generating uniformly distributed values ψ:

$\psi \ \epsilon \ [0, \rho_{d,avg}] \ \rightarrow$ diffuse reflection
$\psi \ \epsilon \ [\rho_{d,avg}, \rho_{d,avg} + \rho_{s,avg}] \ \rightarrow$ specular reflection
$\psi \ \epsilon \ [\rho_{d,avg} + \rho_{s,avg}, 1] \ \rightarrow$ absorption

The photon power will be changed depending on what Russian Roulette will choose as a next action. For instance, if the photon will be diffusely reflected from a certain surface, the ρ_d property of that surface and the incoming flux Φ_i of the photon will be used to compute the power of the reflected photon Φ_r:

$$\Phi_{d,r} = \Phi_{i,r} \frac{\rho_{d,r}}{\rho_{d,avg}}$$

$$\Phi_{d,g} = \Phi_{i,g} \frac{\rho_{d,g}}{\rho_{d,avg}}$$

$$\Phi_{d,b} = \Phi_{i,b} \frac{\rho_{d,b}}{\rho_{d,avg}}$$

The specular reflection and refraction (transmission) are handled in the same way.

3.2 The Photon Map

The photon map is a spatial datastructure through which we can search for a given number of photons near a given point. The choice of datastructure is important in order to perform fast radiance estimation. Like Jensen, we chose a *kd*-tree because this structure is able to locate n photons in $O(log(n))$ time if the tree is balanced. This balancing step is only carried out after all photons have been traced; during tracing they are stored in a flat array for efficiency.

3.3 Radiance Estimation

Estimation of the radiance at a point x is done by exploiting the relationship between radiance and flux [5]. From the Rendering Equation we can express the incoming radiance L_i as follows:

$$L_i(x, \overrightarrow{w}') = \frac{d^2\Phi_i(x, \overrightarrow{w}')}{(\overrightarrow{w}' \cdot \overrightarrow{n_x})d\overrightarrow{w}'dA_i}$$

where the incoming radiance is a function depending on the incoming flux Φ_i over some area A at point x. Substituting this term into the Rendering Equation results in:

$$L_r(x, \overrightarrow{w}) = \int_\Omega f_r(x, \overrightarrow{w}', \overrightarrow{w}) \frac{d^2\Phi_i(x, \overrightarrow{w}')}{dA_i}$$

To approximate the incoming flux, we can use the energy from the photons stored in the photon map. Since each photon in the map has energy $\Delta\Phi_p(\overrightarrow{w}_p)$, we obtain the flux by querying the n photons nearest to the intersection point x within a radius r. The integral can now be rewritten as a sum over photon-energy, and the reflected radiance becomes:

$$L_r(x, \overrightarrow{w}) \approx \sum_{p=1}^{n} f_r(x, \overrightarrow{w}', \overrightarrow{w}) \frac{\Delta\Phi_p(\overrightarrow{w}_p)}{\Delta A}$$

Now that we derived an approximation which can be implemented, we still need to define ΔA. Assume that the closest photons found near point x are all intersecting the same surface as x. Another assumption we need is that the surface around point x is relatively flat. Since we are searching for the nearest photons with respect to point x and within a radius r, this can be seen as a sphere with its center at x which grows until enough photons are found or the maximum radius has been reached. With these assumptions made the area where the photons are located in defines a circle, therefore we can assert $\Delta A = \pi r^2$.

4 Implementation

The implementation for our global illumination algorithm can be described as a two-pass algorithm. In the first pass the photons are emitted from the light sources and have interaction with the scenery to build up the photon map. In our implementation, we simplified the light sources by giving them uniform emission into all directions, with an optional restraint to set a field of view and a direction for this for the lightsources independently. We define light-sources as an array of non-objects, separating them from the other primitives in the scene, resulting in faster intersection tests.

The amount of photons to emit in the first pass is defined by the user in the scene, and the photon map will have an initial size equal to this value. After the map is initialized, the photons are emitted from each light source using the Russian Roulette approach as described in section . Because of the interactions with diffuse surfaces, the number of photons grow larger than the initial map can store and it is important to have the datastructure dynamically. During the first pass, photons can trace light-paths according

to the expression `L(S|D)*D`. This expression captures multiple diffuse reflections which are needed to simulate the illumination effect of color bleeding.

In the second pass of the algorithm, the image is built up by shooting rays into the scene from the camera using the standard ray-tracer and approximating the radiance at intersection points. In algorithm 1, a simplified pseudo-algorithm of our implementation is described where we omit parameters for different illumination calculations. An example of different illumination calculations is to query the photon map for both direct- and indirect illumination, or to employ ray-tracing for the direct component and the photon map for indirect illumination. The pseudo-code describes the latter calculation. In the rendering pass, the set of light-path expressions is captured by `LD?S*E`, the paths that are observed directly or through one or more specular reflections.

An optional technique to estimate indirect illumination which we have implemented is called final gathering [3]. Here, we assume that the distribution of directions incident from the hemisphere above point x is uniform, and sample this distribution repeatedly. A ray is spawned into each sampled direction, and the radiance is estimated at the point at which it intersects the scene from the photon map. Note that there is no recursion at the intersection point to further estimate indirect illumination.

5 Solving the rendering equation

To see if our implementation solves the rendering equation, the equation will be dissected into different parts each concerning an aspect of global illumination. Leaving out time and wavelength, the simplified rendering equation can be written as:

$$L_o(x, \overrightarrow{w}) = L_e(x, \overrightarrow{w}) + L_r(x, \overrightarrow{w})$$

with

$$L_r(x, \overrightarrow{w}) = \int_\Omega f_r(x, \overrightarrow{w}', \overrightarrow{w}) L_i(x, \overrightarrow{w}')(\overrightarrow{w}' \cdot \overrightarrow{n}) d\overrightarrow{w}'$$

L_o describes the outgoing radiance at point x in direction $\overrightarrow{w}$ as the sum of the emitted radiance and reflected radiance at point x in direction $\overrightarrow{w}$. The L_e term can be omitted because we do not include surfaces emitting radiance in our implementation. The reflected radiance L_r is an integral over the hemisphere above x and consists of the BRDF, the incoming radiance and a cosine weight component. Since the BRDF is a combination of diffuse and specular reflectance, this function can be rewritten as the sum of its two parts:

$$f_r(x, \overrightarrow{w}', \overrightarrow{w}) = f_{r,S}(x, \overrightarrow{w}', \overrightarrow{w}) + f_{r,D}(x, \overrightarrow{w}', \overrightarrow{w})$$

where $f_{r,D}$ is the diffuse reflectance term and $f_{r,S}$ is the specular reflectance term. The reflected radiance can also be rewritten as the sum of its parts:

$$L_i(x, \overrightarrow{w}') = L_{i,l}(x, \overrightarrow{w}') + L_{i,c}(x, \overrightarrow{w}') + L_{i,d}(x, \overrightarrow{w}')$$

Figure 1: Rendering pass

```
1: function TraceRay(Ray ray, Scene scene, int rayDepth)
2: Vector irr
3: if (rayDepth < MAXDEPTH) then
4:    rayInt ray_int ← scene.GetClosestObject(ray)
5:    irr ← irr+ ShadeRayPM(ray, rayInt, scene, rayDepth)
6: end if
7: return irr
8:
9: function ShadeRayPM(Ray ray, RayInt rayInt, Scene scene, int
   depth)
10: Vector irr
11: Vector irrTemp
12: Obj obj ← rayInt.getObject()
13: Mat mat ← obj.getMaterial()
14: if (mat.IsSpecularlyReflective()) then
15:    irr ← irr+ SampleDirectIllumination(scene, ray, rayInt, rayDepth)
16: end if
17: if (!mat.IsSpecularlyReflective()) then
18:    irr ← irr+ GatherIrradianceEstimate(rayInt, scene)
19: else
20:    Ray reflectRay ← ray.reflect(rayInt)
21:    irrTemp ← irrTemp+ TraceRay(reflectRay, scene, rayDepth + 1)
22:    irrTemp ← irrTemp ∗ mat.GetSpecularReflectiveness()
23:    irr ← irr + irrTemp
24:    if (mat.IsSpecularlyRefractive()) then
25:       Ray refractRay ← ray.refract(rayInt,mat.GetRefractionIndex())

26:       irrTemp ← irrTemp ∗ 0
27:       irrTemp ← irrTemp+ TraceRay(refractRay, scene, rayDepth +
          1)
28:       irrTemp ← irrTemp ∗ mat.GetSpecularRefractiveness()
29:       irr ← irr + irrTemp
30:    end if
31: end if
32: return irr
```

where $L_{i,l}$ is the incoming radiance from light sources, $L_{i,c}$ is the incoming radiance from caustics and $L_{i,d}$ is the incoming radiance from diffuse inter-reflections. Having split up the main components of the reflected radiance, we can express $L_r(x, \overrightarrow{w})$ as four integrals, each representing a part which can be solved with our implementation.

$$L_r(x, \overrightarrow{w}) = \int_\Omega f_r(x, \overrightarrow{w}', \overrightarrow{w}) L_{i,l}(x, \overrightarrow{w}')(\overrightarrow{w}' \cdot \overrightarrow{n}) d\overrightarrow{w}' +$$

$$\int_\Omega f_{r,S}(x, \overrightarrow{w}', \overrightarrow{w})(L_{i,c}(x, \overrightarrow{w}') + L_{i,d}(x, \overrightarrow{w}'))(\overrightarrow{w}' \cdot \overrightarrow{n}) d\overrightarrow{w}' +$$

$$\int_\Omega f_{r,D}(x, \overrightarrow{w}', \overrightarrow{w}) L_{i,c}(x, \overrightarrow{w}')(\overrightarrow{w}' \cdot \overrightarrow{n}) d\overrightarrow{w}' +$$

Figure 2: Final Gathering

```
function GatherIrradianceEstimate(RayInt rayInt, Scene scene)
Obj obj ← rayInt.getObject()
Mat mat ← obj.getMaterial()
Vector irr, est
#if FINALGATHER == TRUE
for sample = 1 to NUM_SAMPLES do
  Vector sampleRayDir ← SetRandomDir()
  Ray sampleRay(rayInt.getPosition(), sampleRayDir)
  RayInt sampleRayInt
  if scene.GetClosestObject(sampleRay,sampleRayIt) ! = NULL) then
    Obj sampleObj ← sampleRayInt.getObject()
    Mat sampleMat ← sampleObj.getMaterial()
    if !sampleMat.isSpecularlyReflective() then
      est ← photonMap.GetIrradianceEstimate(sampleRayInt, searchRadius, searchCount)
      est ← est · sampleMat.GetDiffuseReflectiveness()
      irr ← irr + est
    end if
  end if
end for
#endif
est ← photonMap.GetIrradianceEstimate(rayInt, searchRadius, searchCount)
est ← est · mat.GetDiffuseReflectiveness()
irr ← irr + est
return irr
```

$$\int_{\Omega} f_{r,D}(x, \overrightarrow{w}', \overrightarrow{w}) L_{i,d}(x, \overrightarrow{w}')(\overrightarrow{w}' \cdot \overrightarrow{n}) d\overrightarrow{w}'$$

5.1 Direct Illumination

The first term in the integral describes the direct illumination, illumination coming directly from the light sources, and is considered the most important of the integrals because human eyes are most sensitive to these type of illumination effects:

$$\int_{\Omega} f_{r}(x, \overrightarrow{w}', \overrightarrow{w}) L_{i,l}(x, \overrightarrow{w}')(\overrightarrow{w}' \cdot \overrightarrow{n}) d\overrightarrow{w}'$$

In figure 5.1 we observe the direct illumination from the photon map. As can be seen, it shows dark shadow at the umbra, but the edges of the penumbra are too blurred. Many more photons are needed in the map in order to get a good approximation of the shadow border, and for this reason we consider using standard ray-tracing for direct illumination. However, shadow edges generated by standard ray-tracing are too discrete, and in order to render penumbra regions more accurately, multiple shadow rays are cast from the point of intersection. A drawback is that the number of shadow rays needed for a correct representation of the penumbra is very high and thus costly.

Direct illumination of specular/glossy objects is not taken into account by this integral for reasons stated in the next subsection.

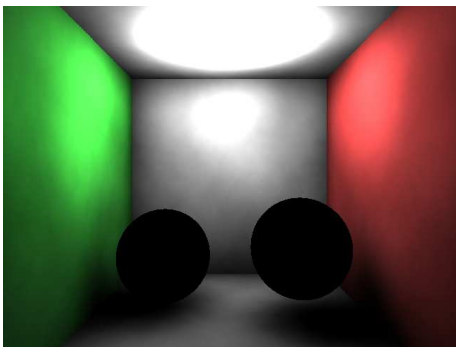

Figure 3: Direct illumination

5.2 Specular and glossy reflections

The second integral describes specular/glossy reflections. We restricted our implementation to handle only perfect specularity, but this effects only the type of reflection model used, not the type of illumination:

$$\int_{\Omega} f_{r,S}(x, \overrightarrow{w}', \overrightarrow{w})(L_{i,c}(x, \overrightarrow{w}') + L_{i,d}(x, \overrightarrow{w}'))(\overrightarrow{w}' \cdot \overrightarrow{n})d\overrightarrow{w}'$$

In this integral, $f_{r,S}$ is dominating as it models the reflection in the mirror direction. Because we only handle perfect specularity, this mirror direction is a very narrow peak in the BRDF. To have a good estimate of this reflection using the photon map, we would need to emit many photons to be able to sample this peak. Because of this, we use the standard ray-tracing to get radiance estimates for specular/glossy surfaces. Figure 5.2 shows the specular reflections only in the cornell box.

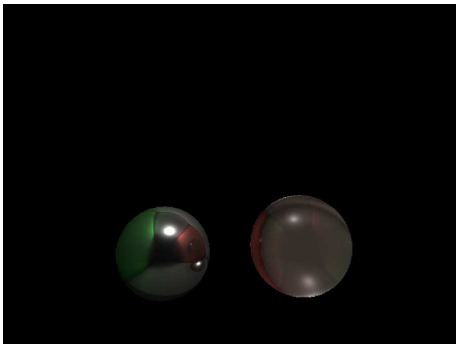

Figure 4: Specular reflection

5.3 Caustics

The third integral describes a form of indirect illumination called caustics:

$$\int_{\Omega} f_{r,D}(x, \overrightarrow{w}', \overrightarrow{w})L_{i,c}(x, \overrightarrow{w}')(\overrightarrow{w}' \cdot \overrightarrow{n})d\overrightarrow{w}'$$

The phenomenom of caustics occur when light gets transmitted through an object with transluscent properties, and the light gets concentrated on another surfaces when leaving the object again. This type of illumination is impossible to render with the standard ray-tracer, and we will use the global illumination map to get good radiance estimations for caustics. In figure 7 we observe the caustic underneath the right sphere. Note that this image displays both caustics and diffuse interreflections because they both are types of indirect illumination.

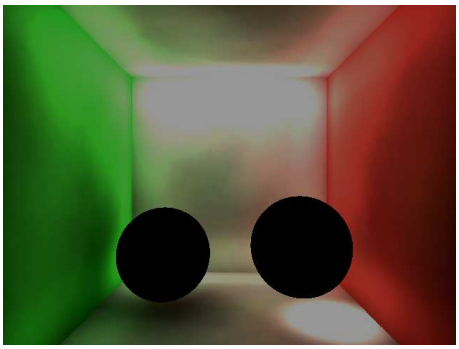

Figure 5: Global illumination

5.4 Diffuse Interreflections

The last integral in the sum accounts for diffuse interreflections, also known as color bleeding:

$$\int_{\Omega} f_{r,D}(x, \overrightarrow{w}', \overrightarrow{w}) L_{i,d}(x, \overrightarrow{w}')(\overrightarrow{w}' \cdot \overrightarrow{n}) d\overrightarrow{w}'$$

Color bleeding is the phenomenom of color exchange between diffuse surfaces. For example, a red carpet near a white wall can reflect photons diffusely on the white wall causing a soft red color to be cast on the white wall. In our implementation we estimate the radiance for diffuse interreflections also using the global illumination map. Note that the white spot on the rear wall is caused by diffuse interreflections, ie. the exchange of colors between the white rear wall and the white ceiling giving a very bright spot on the wall near the light source, and is not a caustic as one could suspect.

6 Results

Various scenes were constructed for our implementation to test on. The resulting images from both standard ray-tracing and photon mapping are shown in the appendix next to each other to demonstrate the difference in quality between the two techniques. The Cornell box is a benchmark to demonstrate the various types of illumination covered by global illumination algorithms, and because of its geometric simplicity it has been used during development. The “recursive box” is an adaption of the Cornell box, demonstrating the possible recursive reflections produced by multiple objects with specular surfaces. The two rooms are demonstrating the importance of indirect illumination when a lightsource is placed outside the room, such that it

scene	exec. time (sec.)	shadow samples	ray-count
Cornell	1.127	0	1.245.188
Cornell SS	39.308	400	234.489.588
recursive	2.108	0	2.199.616
recursive SS	135.088	400	422.350.416
$room_1$	3.522	0	1.551.888
$room_1$ SS	174.809	400	284.641.088
$room_2$	4.912	0	1.637.072
$room_2$ SS	290.854	400	301.825.872

Table 1: Standard ray-tracing profile

scene	exec. time (sec.)	photon init	photon-count
Cornell SS	156.257	250.000	535.004
recursive SS	309.095	250.000	786.937
$room_1$ SS	268.612	2.000.000	1.177.089
$room_2$ SS	290.854	2.000.000	3.066.468

Table 2: Photon mapping profile

provides little direct illumination. Standard ray-tracing cannot render good quality images under these circumstances, whereas photon mapping is still capable of producing high quality images. Table 1 and 2 give an overview on the rendering information for both techniques on the four scenes with various parameters. All images were rendered at a resolution of 800x600 pixels. The maximum recursion depth for the rays are set on 4, and the maximum recursion depth for photon tracing is set on 10.

The photon mapping images are all rendered using 400 shadow ray samples, which give a similar number of ray-counts in the rendering pass as the results of the standard ray-traced scenes with soft shadows (SS) and 2000 photons are used in the estimation.

In the cases of both the rooms, many photons are needed to capture the effect of indirect illumination correctly. Since the light source is outside the room, many photons are emitted into directions with no geometry and therefore not stored in the photon map as is the case with $room_1$. In the case of $room_2$ (an extended version of $room_1$), the extension of geometry causes many diffuse reflections both inside and outside of the room causing the initial number of photons to be doubled in the map.

7 Conclusion

Photon mapping is a powerful addition to standard ray-tracing. The technique can synthesize high-quality images at relatively low computational costs compared to other global illumination methods such as path-tracing.

Because of the decoupling of the statistical illumination information from the scene geometry, it is possible to capture illumination effects such as caustics and diffuse interreflections accurately. However, there are several shortcomings to photon mapping which can lead to artifacts in the results.

When querying the photon map it is posssible that photons can be included from other surfaces that do not belong to the radiance estimation at the intersection point, because a photon does not store which part of the geometry was intersected. There are solutions proposed to this problem, such as applying a compression filter to the spherical search-volume to remove the outliers in the estimation, but these only solve part of the problem. Compression of the volume does not solve the underlying problem with the assumption that surfaces are locally flat when estimating irradiance on a curved surface. Querying the photon map for illumination statistics on a curved surface under this assumption can lead to a highly under-sampled estimation. Another problem with the compression filter is that it does not consider objects intersecting a surface. For example, in figure 7, we demonstrate the indirect light within an entirely closed room. However, because the walls are thin, the estimation near corners and edges on the floor includes photons from outside the room; at these locations the filter does not consider the wall that is seperating indoor from outdoor photons. A possible solution to this problem is to augment the filter with information about the geometrical shapes within its radius or to add different types of filters for different types of surfaces.

Another serious, more general drawback is noise. Being a Monte Carlo method, the sampling error in the photon map decreases proportional to the square root of the number of samples (photons) taken of the function domain (rendering equation) of interest. This means collecting enough statistics to characterize the illumination of non-trivial[7] geometry is expensive, and large variance will be present in the irradiance estimates (recall that photons are adjusted in power by the Russian Roulette procedure). Several measures to reduce noise can be applied, such as importance sampling (in this context, emitting photons in certain preferential directions), irradiance caching (interpolating irradiance values across Lambertian surfaces, on which smooth changes in indirect diffuse illumination are hardest to capture), stratified sampling (preventing photons from becoming non-uniformly distributed), and performing a so-called "final gathering" step (querying the photon map at multiple locations in the hemisphere above a point at which we are estimating the irradiance and averaging). However, most of these either involve a costly precalculation step or increase the run-time complexity of the algorithm to an extent that simply generating extra photons[8] is more effective.

References

[1] James T. Kajiya:
The Rendering Equation, SIGGRAPH 1986: 143, doi:10.1145/15922.15902

[7] ie. scenes that do not consist of many large flat surfaces.

[8] This can essentially be done for free with a properly parallelized implementation.

[2] Arthur Appel:
Some techniques for machine rendering of solids. AFIPS Conference Proc. 32 pp.37-45 (1968)

[3] Matt Pharr, Greg Humphreys: *Physically Based Rendering: From Theory to Implementation, SE p.11 (2010)*

[4] Paul S. Heckbert:
Adaptive Radiosity Textures for Bidirectional Ray Tracing, Computer Graphics Vol. 24 Number 4 August 1990

[5] Henrik Wann Jensen:
Realistic Image Synthesis Using Photon Mapping (2001)

[6] Bui Tuong Phong:
Illumination for Computer Generated Pictures (1975)

Appendix

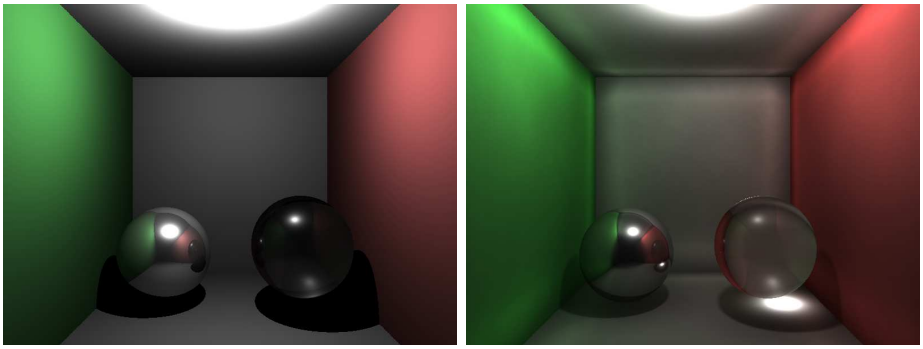

Figure 6: Cornell box. left: ray-tracing, right: photon mapping

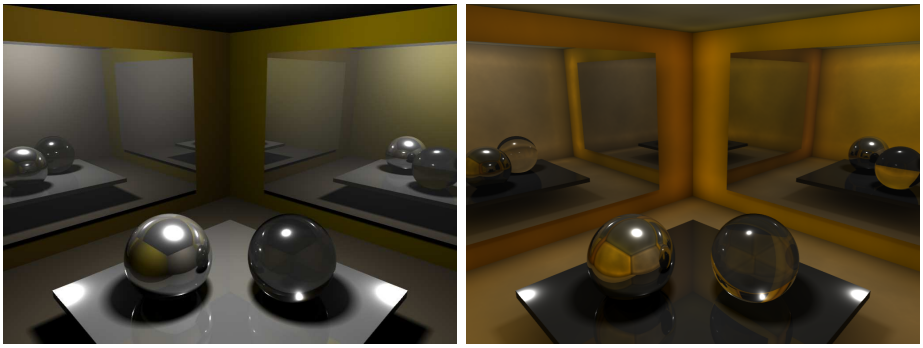

Figure 7: Recursion box. left: ray-tracing, right: photon mapping

Figure 8: Room 1. left: ray-tracing, right: photon mapping

Figure 9: Room 2. left: ray-tracing, right: photon mapping