Advanced Global Illumination and Rendering

Monte Carlo Ray Tracing and Photon Mapping

Algot Sandahl
David Robín Karlsson

December 18, 2020
Linköping University

Abstract

This report outlines the implementation of a multithreaded global illumination renderer based on the Monte Carlo method. To render caustics, the renderer was extended with a simple photon mapping scheme. The renderer supports Lambertian and Oren–Nayar diffuse reflection models, as well as ideal reflective and refracting surfaces. The scene used for testing is a hexagonal room, with a diffuse square light source in the ceiling. In addition, spheres and tetrahedrons were placed in the scene. In the end of the report, renders are presented and the results are discussed.

Contents

1 Introduction

The pursuit of photorealistic renderings has been an elusive goal ever since the dawn of computer graphics. Since the visual perception is based on light, illumination is central to rendering schemes. There exists two main categories of illumination models: *local illumination models* and *global illumination models* which are both heavily used but in different applications.

The fastest illumination models are so called local illumination models, since they operate on a per-object basis. The problem with local illumination models is that since the illumination only depends on the light source and the object itself, the interaction between objects are not taken into account. This means that "effects" like reflections and shadows don't naturally exist in a local illumination rendering. While this can be somewhat remedied by using extensions like shadow mapping and environment maps, these methods are not able to match what a global illumination model can achieve.

To truly achieve realistic renderings, global illumination algorithms based on the physical models of how light interacts in our world are used. In 1986, David S. Immel, Michael F. Cohen and James Kajiya simultaneously developed and encapsulated the now well-known *rendering equation* [1][2]:

$$L_s(x, \Psi_r) = L_e(x, \Psi_r) + \int_\Omega f_r(x, \Psi_i, \Psi_r) L_i(x, \Psi_i) \cos\theta_i \, d\omega_i \tag{1.1}$$

which describes the radiance leaving the point x in the direction Ψ_r as the sum of all reflected and emitted radiance at x. Attempting to solve this equation forms the basis for photorealistic image synthesis. Since the radiance at a single point depends on the incoming radiance from all incoming directions—which in turn has been reflected by other surfaces, and so on—this problem is recursive in nature. This quality makes the rendering equation notably difficult to solve in an accurate and efficient way. Historically the two dominating methods have been *radiosity* and *Whitted ray tracing*.

The Whitted ray tracing scheme was presented in 1980 by Turner Whitted [3] as an extension to previous ray tracing algorithms that did not take reflection or refraction into account. Whitted ray tracing solves the rendering equation by sending importance rays from the eye point through pixels out into the scene. If a ray hits a mirror or a transparent object, one or two new rays are spawned in the perfect reflecting and/or refraction direction. If a ray hits a diffuse object, a ray is sent to the light source to check if the light path is obstructed; if not, a local illumination model is used. While this was an improvement compared to previous efforts, color bleeding—reflections onto diffuse surfaces—cannot be resolved by Whitted ray tracing.

The radiosity rendering scheme was developed in 1984 at Cornell University [4] and Hiroshima University [5] using theory initially developed for engineering related to heat transfer. The radiosity method exploits the fact that under certain assumptions—notably all surfaces being diffuse—the finite element method can be used to solve the rendering equation. Only being able to solve the equation for diffuse surfaces is certainly a big limitation, but radiosity handles color bleeding as opposed to the Whitted ray tracing scheme. A perk with the radiosity method is that it is viewpoint independent, hence it can be used offline in for example game engines to create light maps that can be used in real-time. In summary, radiosity handles diffuse reflections well, but it cannot be used for specular or transparent object which means it is not that useful for general scenes. It can, however, be used in combination with other techniques.

The *Monte Carlo rendering scheme,* introduced in [2] is a stochastic rendering algorithm that resolve many of the problems of earlier algorithms. The idea is to essentially render many images where the ray paths are chosen randomly. The average of all these images will then converge to a photorealistic image as the number of images increases. To make the convergence faster, various techniques such as importance sampling and explicit direct lighting are used. The Monte Carlo scheme is descibed further in Section 2.3.

The main drawback of the Monte Carlo method is that caustics cannot be produced using reasonable amounts of computer resources. In 1996 Wann Jensen presented a two-pass algorithm that can be used to render caustics [6]. In the first pass, packets of flux, called "photons", are emitted from the light out into the scene and as they hit diffuse surfaces, they are put into a *photon map*. The photons are then reflected and refracted in a similar way as the rays in Monte Carlo ray tracing. The second pass consists of a distributed ray tracing algorithm that use the photon maps to render caustics as well as optimize the rendering. To do this the incoming radiance in (1.1) is split into different parts depending on how and if it has been reflected since the light source. Then direct illumination, specular reflections, soft illumination and caustics can be treated separately in an effective manner.

The presented solution uses two photon maps: a *caustic photon map* and a *global photon map*. The global photon map has a lower resolution and is not visualized directly, but is instead used as an estimate for the soft indirect illumination after one or a few reflections. The caustic photon map is used for the caustics in the scene and need to be visualized directly, so it demands a higher resolution. In addition to this, *shadow photons*, are placed at all surface intersections of emitted photons, except the first one, indicating that the path to the light source is blocked. These can then be used to evaluate the visibility term in the direct light term in many cases, avoiding a large amount of shadow rays.

This paper covers the background and implementation of a Monte Carlo renderer to approximate the rendering equation. To render caustics, a simplified photon mapping scheme is implemented. The main objective of this alternative photon mapping approach is to see if Wann Jensen's approach can be simplified, considering the technical advancements made in computer hardware since his paper was published.

2 Background

Our renderer is a Monte Carlo renderer, extended with a simple version of photon mapping to be able to render caustics. The theoretical models that lay the foundation for the renderer is described in this chapter.

2.1 Bidirectional Reflectance Distribution Function

When a ray of light hits the surface of an object the properties of this surface determine what happens to the ray. The ray of light could for example be reflected or scattered depending on what kind of surface the ray hit. These surface properties needs to be considered when synthesizing photorealistic images and they are described mathematically with a *bidirectional reflectance distribution function*, often simply referred to as a BRDF.

$$f_r(x, \omega_{in}, \omega_{out}) \tag{2.1}$$

Equation (2.1) shows the general form the BRDF function takes. This general BRDF form can also be seen in the rendering equation, (1.1), albeit with differently named parameters. Unfortunately it's impossible to mathematically represent surface properties completely accurately. Fortunately, accurate models have been developed that can be used for photo realistic image synthesis.

2.1.1 Lambertian Model

The Lambertian reflectance model is modeled after a matte surface. It's a simple model where the BRDF function f_r is constant for all directions and is given by

$$f_r(x, \omega_{in}, \omega_{out}) = f_r(x) = \frac{\rho}{\pi} \tag{2.2}$$

where ρ is the reflection coefficient *albedo* and π is used as a normalization factor. Equation (2.2) is a simplification of the general BRDF form as demonstrated in (2.1) as it omits the parameters $\omega_{in}, \omega_{out}$ which means its independent from whichever way light falls upon the surface.

2.1.2 Oren–Nayar Model

The Oren–Nayar model is modeled after rough surfaces and is significantly more complicated than the Lambertian model. The model achieves this rough surface by assuming

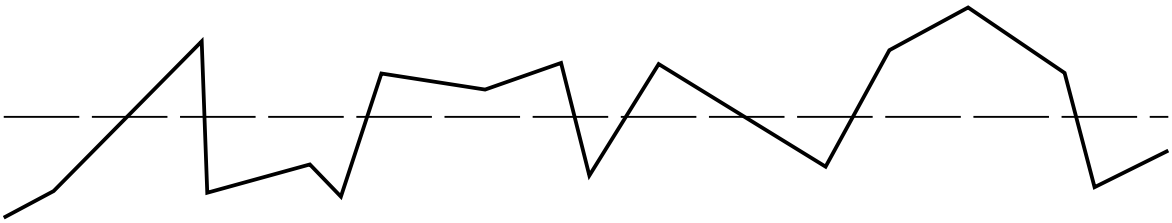


Figure 2.1: *2D representation of a microfacet surface where the dotted line is the average level of the surface.*

that the surface consists of V-shaped microfacets which are Lambertian reflectors themselves.

Every microfacet normal forms an angle with the average normal of the surface which can be realized by looking at Figure 2.1. However, defining a normal for every microfacet is difficult. The normal of every microfacet can instead be modeled statistically with a Gaussian distibution which simplifies this problem into straightforward calculations. The variance of the distribution σ^2 represents how rough the surface is and is utilized in the helper variables

$$A = 1 - 0.5\frac{\sigma^2}{\sigma^2 + 0.33}, \quad B = 0.45\frac{\sigma^2}{\sigma^2 + 0.09}. \tag{2.3}$$

The variables A and B in equation 2.3 is then utilized in the BRDF like

$$f_r(x, \omega_{in}, \omega_{out}) = \frac{\rho}{\pi} \cdot (A + (B \cdot \max[0, \cos(\phi_i - \phi_r)] \sin\alpha \tan\beta) \tag{2.4}$$

where ρ is the albedo of the surface, θ_i, ϕ_i and θ_r, ϕ_r are the inclination and azimuth angles of incoming and exiting rays $\omega_{in}, \omega_{out}$, $\alpha = \max[\theta_i, \theta_r]$ and $\beta = \min[\phi_i, \phi_r]$. As (2.4) demonstrates this model revolves heavily around the direction of the incoming and outgoing rays $\omega_{in}, \omega_{out}$ and the roughness σ^2 combined with an albedo ρ to compute the BRDF.

One interesting observation is that if $\sigma = 0$ then $A = 1$, $B = 0$ and (2.4) simplifies to the Lambertian model, (2.2), reinforcing that a Lambertian surface exhibits no roughness.

2.2 Reflection and Refraction

Pefect specular reflections and refraction can be efficiently handled by the Whitted ray tracing scheme. The renderer uses this method to calculate reflections and refractions. If a light ray intersects a perfect mirror, a new ray with the same radiance/importance is sent in the perfect reflection direction, which is given by

$$\mathbf{R} = \mathbf{I} - 2(\mathbf{I} \cdot \mathbf{N})\mathbf{N} \tag{2.5}$$

where $\mathbf{I}$ is the normalized direction of the incoming ray and $\mathbf{N}$ is the surface normal.

If a transparent object is intersected, two new rays are spawned: one transmitted ray, and one reflected ray. The incoming radiance is split between these rays using Schlick's approximation of the computationally expensive Fresnel formula. Schlick's approximation for the reflection coefficient is given by

$$R(\theta) = R_0 + (1 - R_0)(1 - \cos\theta)^5, \quad R_0 = \left(\frac{n_1 - n_2}{n_1 + n_2}\right)^2 \tag{2.6}$$

where θ is the angle between the incoming direction and the surface normal. n_1 and n_2 are the refractive indices of the two media. A reflected ray with R times the incoming radiance/importance is sent in the perfect reflection direction calculated by (2.5), the remaining radiance is put in to the transmitted ray whose direction is given by

$$\mathbf{T} = \frac{n_1}{n_2}\mathbf{I} + \mathbf{N}\left(-\frac{n_1}{n_2}[\mathbf{N}\cdot\mathbf{I}] - \sqrt{1 - \left[\frac{n_1}{n_2}\right]^2 [1 - (\mathbf{N}\cdot\mathbf{I})^2]}\right) \tag{2.7}$$

Care has to be taken when $n_1 > n_2$, since total reflection occurs for θ bigger than the brewster angle:

$$\theta_B = \arctan\frac{n_2}{n_1}. \tag{2.8}$$

2.3 Monte Carlo Ray Tracing

In Monte Carlo ray tracing, rays carrying *importance*, which behaves identically to radiance but travels in the opposite direction, are sent out from a camera into the scene similarly to Whitted ray tracing. In real life, when rays of light intersects a diffuse surface the light is scattered in an infinite amount of hemispherically bounded directions. The Monte Carlo method employs *path tracing* which simulates how these infinitely reflected rays of light could have ended up in the camera. Naturally, it's difficult to integrate over an infinite amount of incoming directions, therefore Monte Carlo raytracing uses a Monte Carlo algorithm to to approximate the radiance at a point x in the direction ω_{out} as

$$\begin{aligned} L(x \to \omega_{out}) &= \int_\Omega f_r(x, \omega_{in}, \omega_{out}) L(x \leftarrow \omega_{in}) \\ &\approx \frac{\pi}{N} \sum_{i=1}^{N} f_r(x, \omega_{in}, \omega_{out}) L(x \leftarrow \omega_{in}) \end{aligned} \tag{2.9}$$

where N is the number of randomly generated rays to be cast. Each generated ray sent out into the scene needs to follow a set of rules to adhere to photorealism. The rays are realistically reflected and refracted as discussed in Section 2.2. If a generated ray intersects a light source it's terminated. The problem arises when the generated ray intersects a diffuse surface. If the Monte Carlo method is used on this intersected surface the time needed to render the image would increase drastically, bordering on infinite. Therefore we need to use a mechanism to terminate generated rays on diffuse surfaces.

To uphold the promise of converging towards photorealism the termination method needs to be unbiased, otherwise unnatural artifacts could form in the rendered image. The termination scheme used is called *Russian roulette*.

2.3.1 Russian Roulette

The Monte Carlo method uses Russian roulette as its ray termination scheme. Just like real Russian roulette, this scheme is completely based on probabilistic outcomes which are inherently natural. This makes Russian roulette an unbiased scheme for terminating rays which allows the renderer to converge towards photorealism.

Whenever a ray generated by the Monte Carlo method intersects a diffuse surface, a uniformly random generated number $t \in [0, 1]$ is generated. This number is examined together with a static ray absorption probability α. If $t > 1 - \alpha$ the ray is terminated. Otherwise another ray is randomly generated at the intersection point which is cast into the scene using the same ray termination scheme.

This scheme relies on a good pseudo-random number generator for t be truly unbiased. Luckily, modern pseudo-random number generator are highly sophisticated and can be used in this scheme. However, this scheme presents a problem in its current form. Let's consider a more general case: suppose that we have a Monte Carlo algorithm based estimator that randomly generates N instances of x used in an arbitrary function $f(x)$ like

$$I = \frac{1}{N} \sum_{i=1}^{N} f(x_i)$$

where the expected value of I is $E[I] = \lambda$. We combine I with a Russian roulette discard rate α, similar to the absorption rate used earlier. Then we can approximate the total number of non discarded values from $f(x)$ as

$$\frac{1}{N} \sum_{i=1}^{N(1-\alpha)} f(x_i) = (1-\alpha)\frac{N}{N} f_{i...N}(x_{i...N}) = (1-\alpha)\lambda \tag{2.10}$$

We can see in (2.10) that discarding α values from the estimator creates a bias $1 - \alpha$ in the end result. This bias needs to be removed before we can employ such an estimator in our renderer. This done by simply adding the denominator $P = 1 - \alpha$ to the estimator in (2.10):

$$\frac{1}{PN} \sum_{i=1}^{N(1-\alpha)} f(x_i). \tag{2.11}$$

This estimator is used in the renderer to make unbiased Monte Carlo estimations of diffusely reflected light.

2.3.2 Direct Light Contribution

Since the light source is typically only a small part of the scene, most rays will be terminated before ever reaching the light; considering that the light is typically the only

emitting object in a scene, these rays will contribute nothing to the image. This makes the convergence very slow. To obtain good images faster, light directly from the light source can be explicitly taken into account at each intersection point, using the Monte Carlo method by sending M shadow rays to random points, q_k, on the light. [7] The direct light contribution from one light at a point x in the direction ω is estimated by

$$\langle L_D(x \to \omega) \rangle = \frac{AL_0}{M} \sum_{k=1}^{M} f_r(x, \omega_{in,k}, \omega) \frac{\cos \alpha_k \cos \beta_k}{d_k^2} V_k \tag{2.12}$$

where A is the area of the light source, L_0 is the radiance leaving the light at any point in any direction, α_k is the angle between the light normal and the shadow ray, β_k is the angle between the surface normal and the shadow ray and d_k is the distance between q_k and x. V_k is a visibility term and is 1 if nothing is blocking the shadow ray, else $V_k = 0$.

2.4 Anti-Aliasing

In renderings, the rendered scene gets represented using a limited resolution. This can introduce aliasing, appearing for example as jagged lines. To alleviate this problem, the point at which each ray intersects the pixel on the view plane can be randomized. Given that multiple samples are used per pixel, this will produce a smoother image.

2.5 Photon Mapping

Photon mapping is a method developed by W. Jensen in 1995 [8]. The purpose of photon mapping is to approximate the rendering equation, (1.1), using a two-pass method. The objective of the first pass is to realistically simulate how light sources emit light onto objects in a scene. This is achieved by casting a lot of photons carrying small amounts of flux, similar to real photons from which the method derives its named. The rays are cast in hemispherically bounded random direction originating from random points on a light source's surface. Each ray is cast into the scene with a path tracing algorithm that supports realistic reflection and refraction as described in Section 2.2 on specular surfaces and diffuse reflection using Russian roulette as described in Section 2.3.1. Each ray eventually terminates on a diffuse surface and the point where the ray terminated along with its incoming direction and its flux is stored in the photon map. The same is done for each "bounce" on a diffuse surface.

During the second pass, which consists of a typical ray tracer, this information is used to estimate the radiance leaving a point, x, in the direction ω according to

$$L(x \to \omega) \approx \sum_{i=1}^{n} f_r(x, \omega, \omega_i) \frac{\Delta \Phi_i(x \leftarrow \omega_i)}{\Delta A} \tag{2.13}$$

where n is the number of photons within a predetermined search range, f_r is the BRDF as before, $\Delta \Phi_i(x \leftarrow \omega_i)$ is the flux arriving at x from direction ω_i from retrieved photon number i, ω_i is the incoming direction of retrieved photon number i, and ΔA is

the estimated area which the estimate is done over, and is determined by the search range.

When compared to Wann Jensen's method we make some significant simplifications: only one photon map is used, and the estimates given by (2.13) are used also for the first reflections—this is necessary since we do not use a separate caustic map.

2.6 Post-Processing

Since the renderer works with physical units, some processing is needed before creating an image. A curve is applied to the normalized irradiance values of each pixel, after clipping the highest values to achieve a good exposure. These are then normalized to $[0, 255]$ and written to a PNG.

2.7 The Scene

The scene used for testing is depicted in Figure 2.2 and 2.3 with camera plane and eye points visualized. As this Monte Carlo renderer is based around terminating rays on diffuse surfaces the camera plane and eye points are fully contained inside the room to make sure no ray escape its boundaries. Two different eye points can be used to render the image which results in different perspectives in the final image.

2.8 Implementation

The renderer was implemented from scratch in C++ and uses *GLM, OpenGL Mathematics*, for fast vector calculations. To write the result to a PNG, the library *LodePNG* is used. To lower the rendering times the renderer uses multithreading functionality from the C++ standard library. The photon map was implemented using a k-d tree provided by the library *libkdtree++*. The renderer was run on machines well suited for multithreading, the most powerful having 16 logical cores and an impressive 128 GB RAM memory which enabled the use of large photon maps.

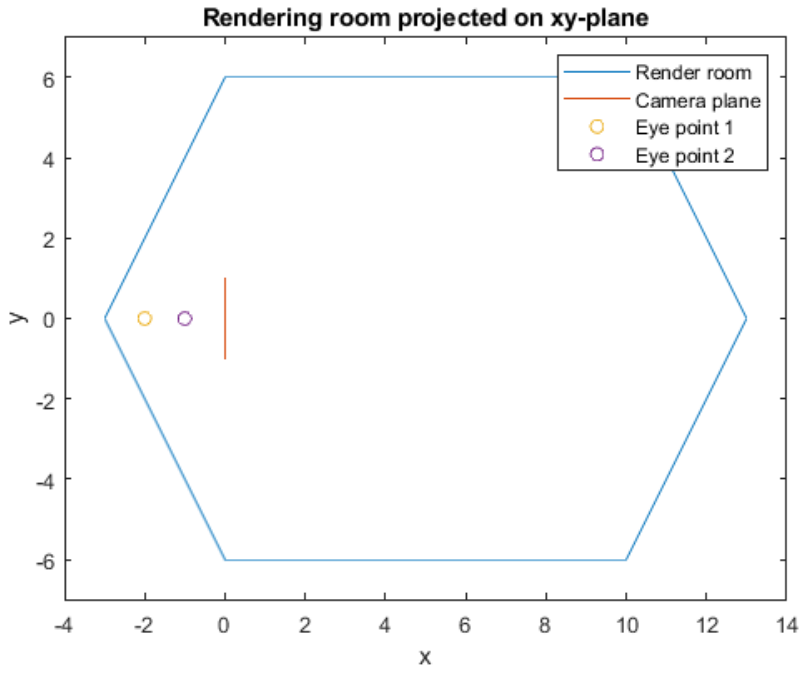


Figure 2.2: *The rendering room projected on the xy-plane, displaying the shape of the floor and ceiling. Both eye points and camera plane are visible.*

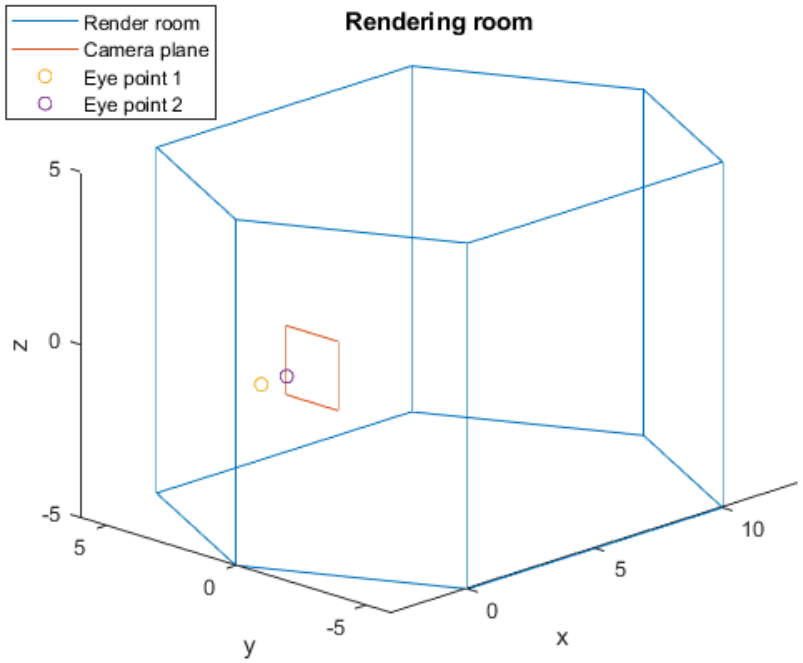


Figure 2.3: *The render room in 3D. Both eye points and camera plane are visible*

3 Results

This chapter presents rendered images, comparisons and tables detailing rendering parameters.

3.1 Monte Carlo Ray Tracing

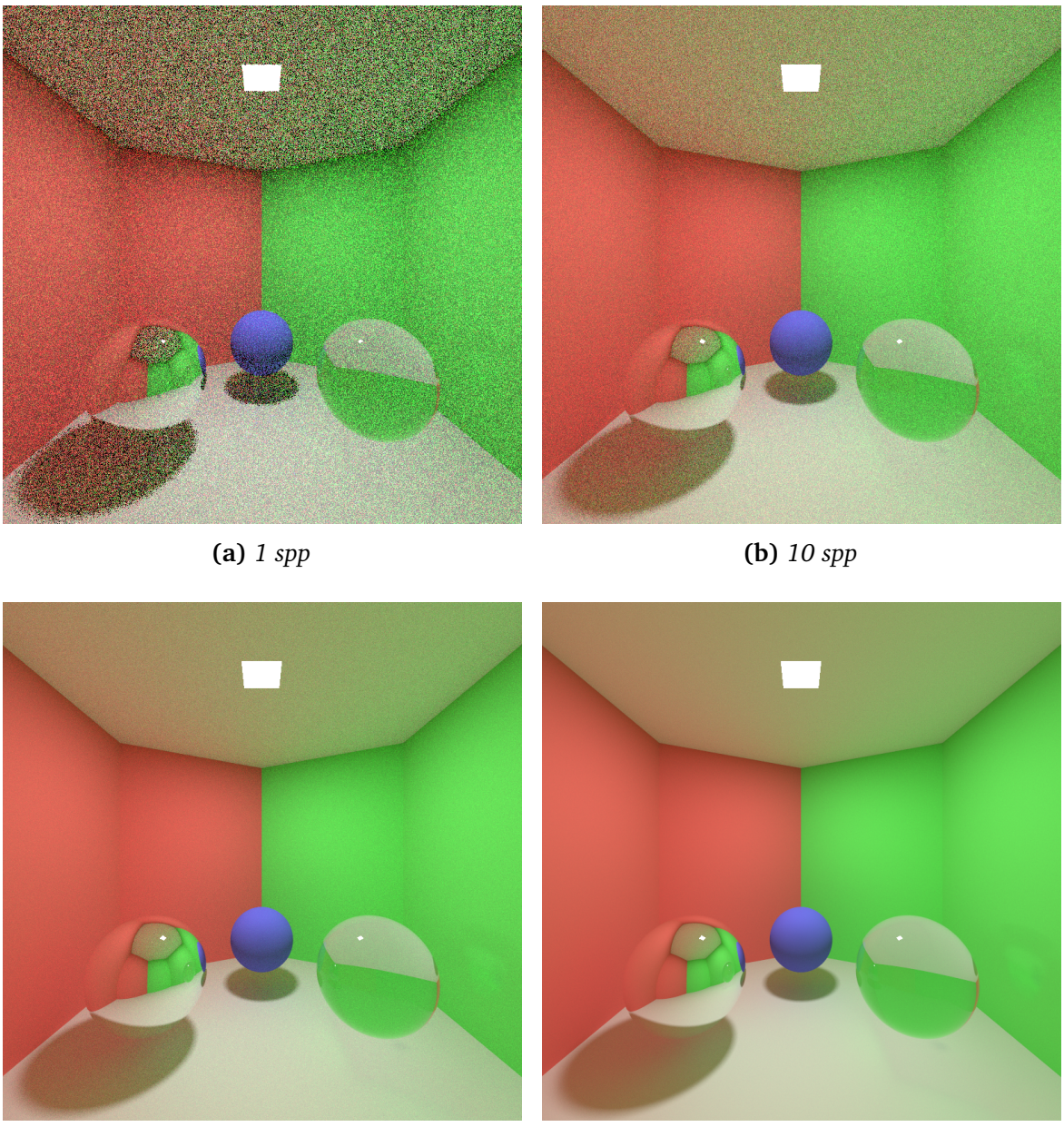
Renderings produced by the Monte Carlo method without photon mapping can be seen in Figure 3.1. The images were rendered with different amounts of samples per pixel (spp), but all with the resolution 800×800 pixels, and the termination probability 0.2. The rendering times can be seen in Table 3.1 and were measured on an AMD Ryzen 5 2600X six-core computer with 16 GB RAM. As expected the rendering time is roughly proportional to the spp—the noise, however, does not decrease as drastically.

Table 3.1: *The rendering times for the Monte Carlo renders seen in Figure 3.1*

Spp	Rendering time
1	00h:00m:06s
10	00h:00m:50s
100	00h:08m:15s
1,000	01h:20m:14s

From these renders it becomes evident that this method does not handle the glass sphere in a way that is realistic. This comes from the simplification that the shadow rays used for the explicit light contribution do not consider transparent objects. If the direct lighting model is not used, and radiance is instead fed in only through rays hitting the light source, caustics do appear, as can be seen in Figure 3.2a. The problem with this approach is that the amount of samples needed per pixel to achieve an image with a reasonable amount of noise is immense and it is therefore unfeasible, as becomes evident by comparing Figure 3.1c and 3.2a which were both rendered using a hundred samples per pixel.

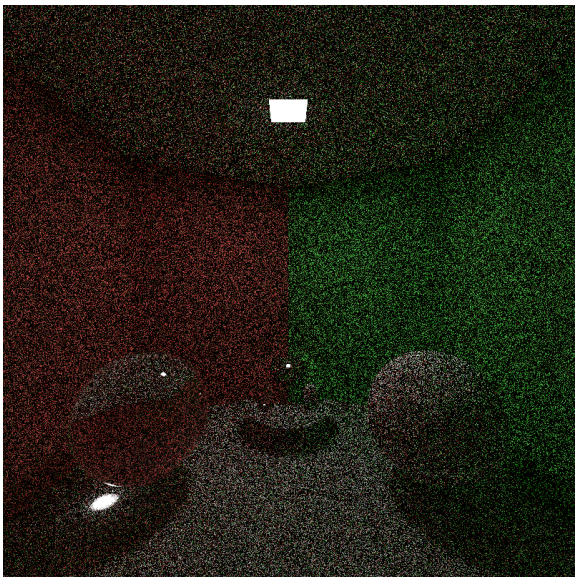
A scene containing a tetrahedron can be seen in Figure 3.2b. The shadow casted by the tetrahedron demonstrates how the shadow when using an area light source is sharper when the distance between the object and the shadow is small, and blurrier as the distance become greater.



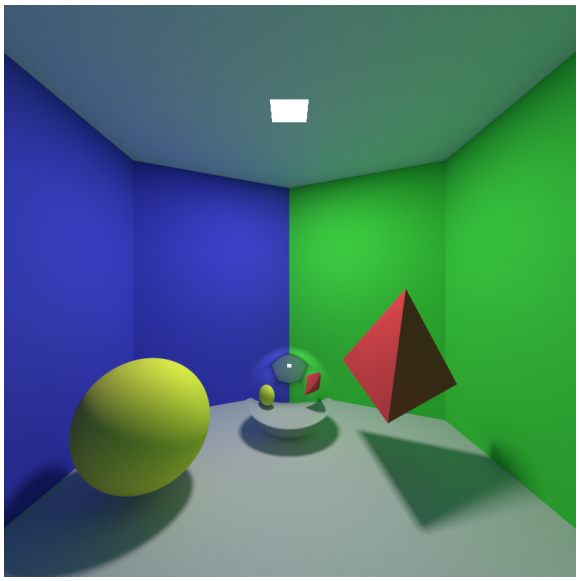
(a) *1 spp* **(b)** *10 spp*

(c) *100 spp* **(d)** *1,000 spp*

Figure 3.1: *Monte Carlo renders using different amount of samples per pixel.*



(a) *An image rendered with Monte Carlo ray tracing, without the direct light model. A hundred samples were used per pixel.*

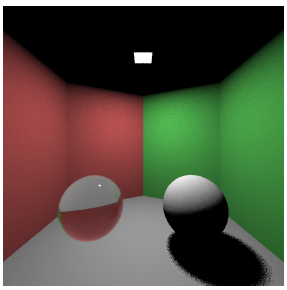


(b) *A Monte Carlo render with a yellow sphere, a mirror ball and a red tetrahedron. A thousand samples were used per pixel.*

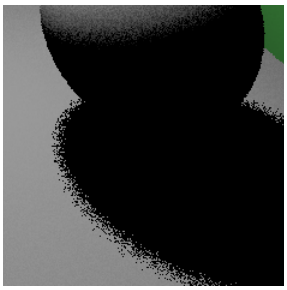
Figure 3.2: *Monte Carlo renderings.*

3.1.1 Direct Light Contribution

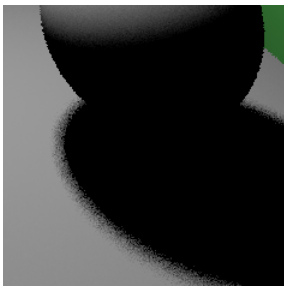
In Figure 3.3 only the direct light contribution is taken into account on the diffuse surfaces, essentially making the renderer into a Whitted ray tracer (with some exceptions, primarily in the handling of shadow rays). Figure 3.3a clearly shows the problem with the Whitted ray tracing scheme: all diffuse surfaces that are not directly illuminated by the light source become completely black (unless an ambient term is used), see for example the ceiling, or the shadow behind the right sphere. In reality these areas would be significantly illuminated by reflections, as seen in the renders produced by the full Monte Carlo scheme.



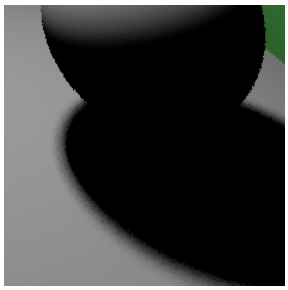
(a) *1 shadow ray*



(b) *1 shadow ray*



(c) *10 shadow rays*



(d) *100 shadow rays*

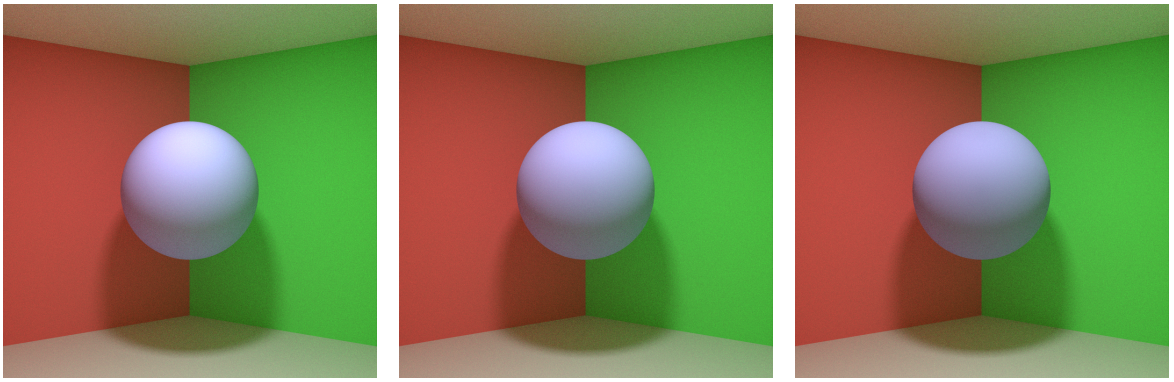
Figure 3.3: *Only direct illumination, 1 spp, using different amount of shadow rays.*

Figure 3.3 also shows a comparison of different amount of shadow rays, M, used in the estimator (2.12). Clearly more rays produce smoother shadows, as is to be expected.

When using the direct light estimator with the full Monte Carlo scheme, however, $M = 1$ is sufficient given a high enough spp, since each pixel is sampled many times so the final pixel is still estimated using multiple shadow rays. Since the spp has to be high to reduce the noise, this is essentially always the case.

3.1.2 Oren–Nayar Roughness Comparisons

Different values of the roughness, σ, of the Oren–Nayar BRDF are demonstrated by a sphere in Figure 3.4.

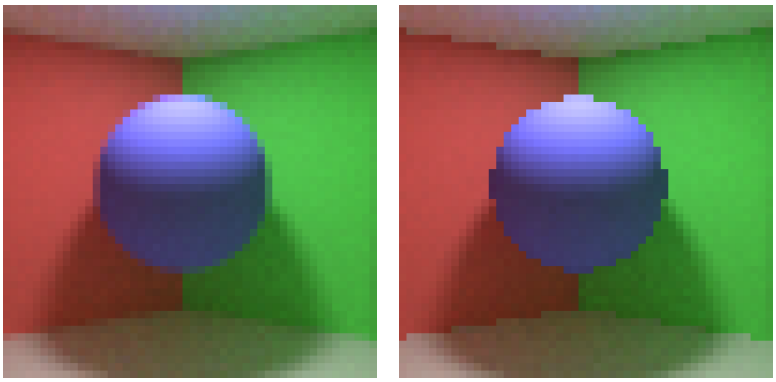


(a) $\sigma = 0$ **(b)** $\sigma = 0.5$ **(c)** $\sigma = 1$

Figure 3.4: *A sphere rendered using the Oren–Nayar model with various roughness values.*

3.1.3 Anti-Aliasing

The effect of the ray randomization used for anti-aliasing can be observed in Figure 3.5, where a scene has been rendered using a very low resolution. As seen the edges look more natural and smoother when ray randomization is used.



(a) *Ray randomization* **(b)** *No anti-aliasing*

Figure 3.5: *Anti-aliasing comparison, 1000 spp*

3.2 Photon Mapping

Figure 3.6 shows photon mapping renderings using different search ranges, but all with 9,000,000 photons emitted by the light source. All renderings are in the resolution 800×800 pixels. The amount of photons in the photon map when pass 1 was finished, among other information, can be seen in Table 3.2. The rendering times were measured on an AMD Ryzen 5 2600X six-core computer with 16 GB RAM. The time needed for pass 2 increases drastically as the search range is increased, which is explained by the increased amount of photons to find in the photon map.

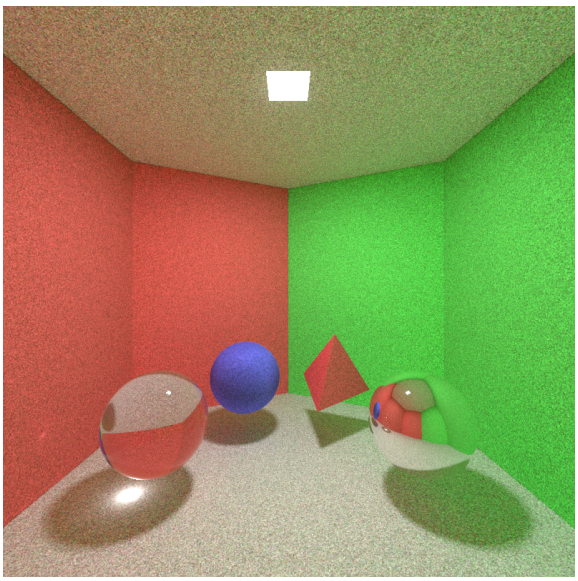
Table 3.2: *Rendering times and parameters for some photon mapping renders*

Figure	Amount of photons	Search range	Spp	Pass 1	Pass 2
3.6a	51,008,900	0.01	10	00h:04m:22s	00h:04m:36s
3.6b	51,035,563	0.05	10	00h:04m:05s	00h:11m:31s
3.6c	50,994,129	0.10	10	00h:04m:29s	01h:32m:17s
3.7b	47,735,650	0.05	10	00h:04m:27s	01h:11m:37s

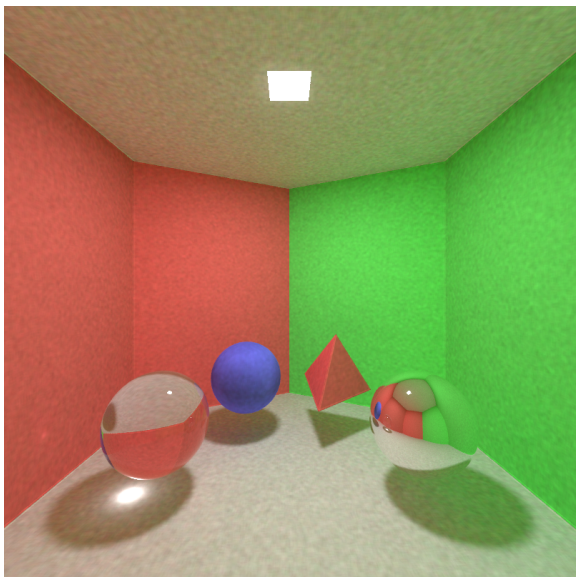
Figure 3.6 also shows that larger search ranges produce smoother images when using the same amount of photons, but since the search is done in three dimensions even though the aim is to find photons on a surface, artifacts are introduced along edges, as seen in Figure 3.6d.

Figure 3.7b shows an example of a scene that would not be properly rendered without the photon mapping. Note especially the double shadows casted by the blue sphere: one casted by the light coming directly from the light source, this shadow is correctly produced also in Monte Carlo rendering (Figure 3.7a); the other shadow is from the light reflected by the mirror, and is not present in the Monte Carlo rendering.

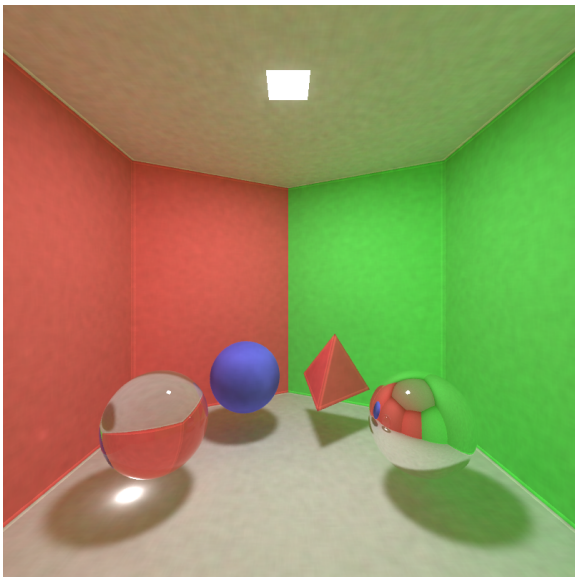
Figure 3.8 shows a rendering produced with a very high photon count. Unfortunately there was an error during rendering that produced bright artifacts. This image also use Monte Carlo ray tracing in areas where shadow photons are present, as opposed to Figure 3.7 and 3.6 which use the photon maps also for shadows. Since Monte Carlo is used for the shadow regions, a higher spp is needed when compared to rendering all diffuse areas using the photon map. In the latter case, multiple samples are used primarily for the sake of anti-aliasing.



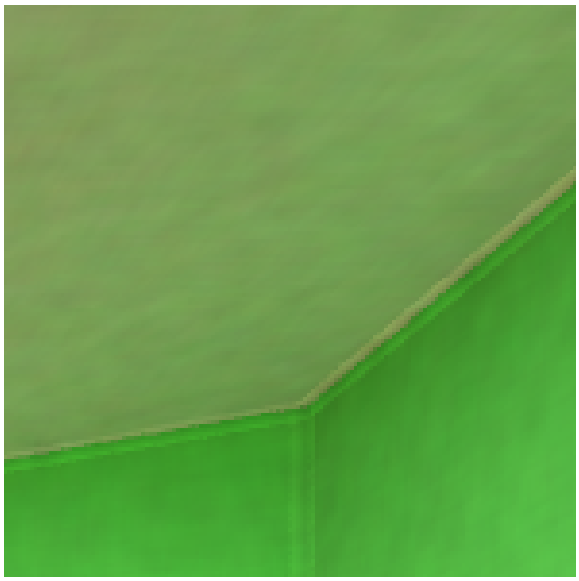
(a) *Search range 0.01*



(b) *Search range 0.05*

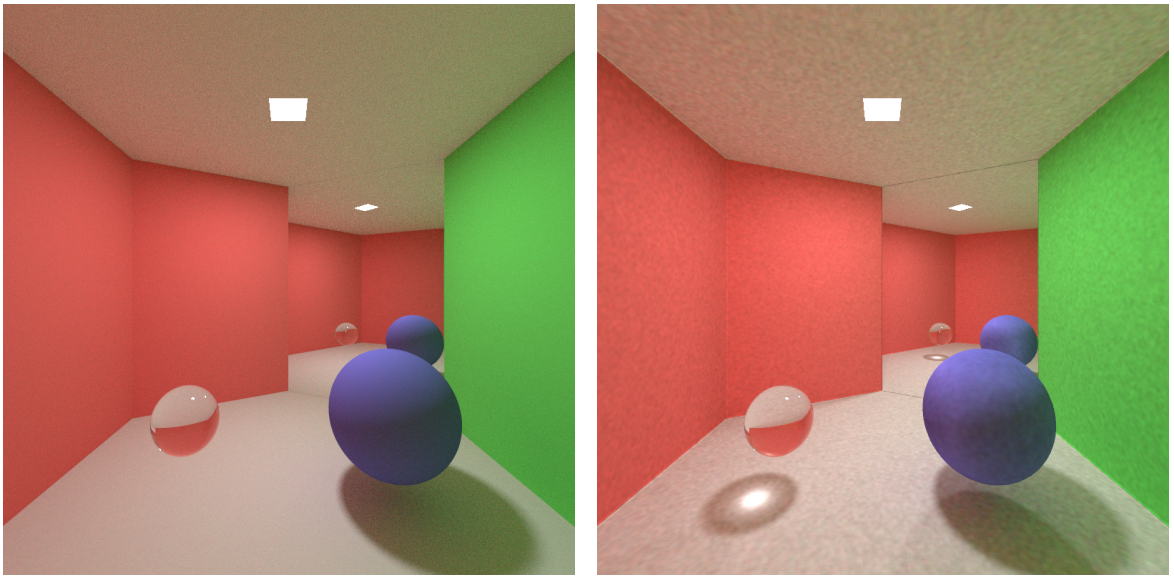


(c) *Search range 0.1*



(d) *Close-up of Figure 3.6c*

Figure 3.6: *Photon map renderings, with different search ranges.*



(a) *Monte Carlo* **(b)** *Photon mapping*

Figure 3.7: *A scene including a mirror wall rendered with Monte Carlo and photon mapping.*

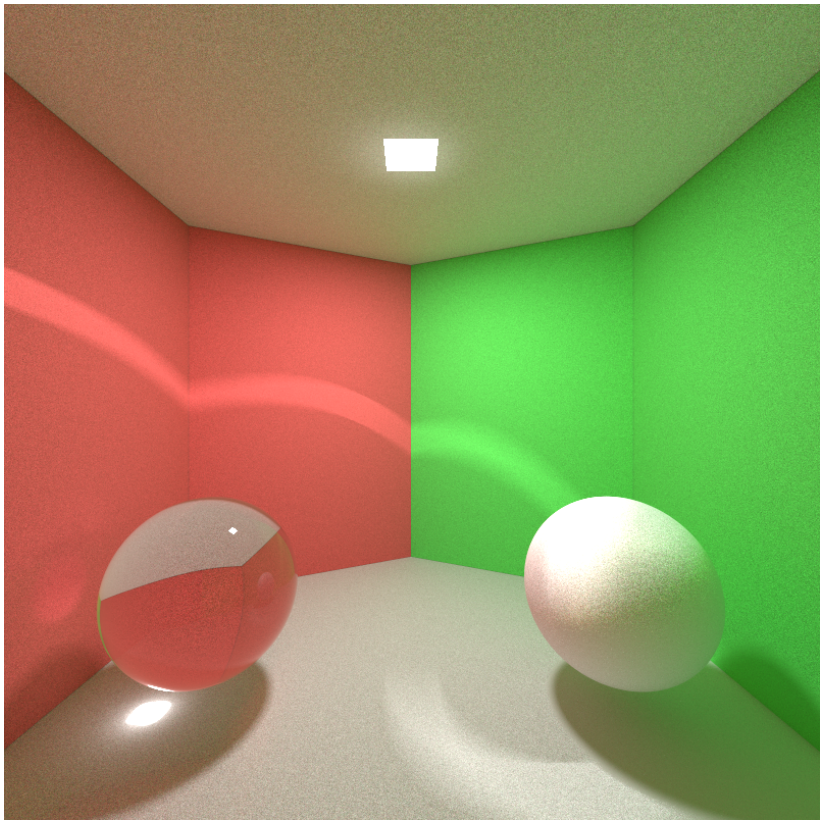


Figure 3.8: *A scene including a transparent sphere and a lambertian diffuse reflector rendered on the most powerful computer with photon mapping and monte carlo path tracing. The photonmap contained 614,971,808 photons and opccupied 79 GB of memory. Monte Carlo ray tracing was used for regions that contained shadow photons.*

4 Discussion

Implementing a global illumination renderer is a task that requires a lot of knowledge as well as time. It is not enough that the renderer produces correct results—it also needs to be reasonably fast. As demonstrated in Chapter 3, obtaining a pleasing image requires long rendering times. The fact that the presented renders in this report only contain a few objects needs to be taken into account. In real-world applications, such as movies, the amount of objects in a scene is highly substantial. This calls for ways of storing the scene in which a ray–object intersection test is not needed between every ray and object in the scene.

The use of multithreading speeds up the rendering times considerably, and since almost all computers today have multiple threads it is almost a necessity if rendering times is of any concern. Luckily the standard ray tracing scheme (inluding Monte Carlo) is well suited for threading since each ray originating from the camera is completely independent from all other rays. The only shared data is the scene geometry, which is constant throughout the duration of the rendering and thus safe for multiple threads to read from. Multithreading is also used in the construction of the photon map although it is not as straightforward as with the rendering as the threads will modify the same data structure representing the photon map. The construction of the photon map became faster but the gains were not as substantial as with the rendering.

To produce convincing renders of glass objects, caustics need to be present. To efficiently render these using Monte Carlo presents challenges since it is a phenomenon that is highly dependent on the light. To remedy this, a light-driven photon map can be used to complement the eye-driven Monte Carlo rendering step. Thus the direct light falling behind the glass sphere been refracted by the glass, resulting in caustics.

The simplified photon mapping scheme used in the renderer proved to be very demanding in terms of memory requirements, since the amount of photons needed to produce an image without the low-frequency noise present in for example Figure 3.7b or the artifacts along edges seen in Figure 3.6d, is very high. When using an amount of photons that high, the role of shadow photons also become questionable, since the photon map resolution needed to make the other parts of the image look good means that also the shadow regions can be estimated using the photon map. Given the memory requirements of the simplified method, and the fact that the more photons used, the slower the look-up is, using separate photon maps as described in [6] still seems like the better choice.

Large software projects are never free of bugs. This is evident in Figure 3.8 where curved light patterns appears in the final image. This bug occurs somewhere in the processes involving refraction and reflection with transparent surfaces. A large amount of time

has been put into attempts to resolve it but the bug still persists. This particular bug is hard to resolve as it only appears when using large photon maps, whose construction is a lengthy process. This means that rendering an image to see if the bug was resolved took around seven hours each attempt, making this bug extremely difficult to fix.

There are also various ways to improve the final image through post-processing. Noise reduction is probably one of the most useful forms of post-processing; for example the rendering times required for Monte Carlo renderings can be lowered if some noise can be removed in post. Since the images are rendered with a static exposure, the dynamic range is not automatically optimized to the scene. To heighten the dynamic range, auto-exposure and various tone mapping techniques could be applied, transforming the image into something more similar to how our eyes, or a camera, would perceive the scene.

Bibliography

[1] D. S. Immel, M. F. Cohen, and D. P. Greenberg, "A radiosity method for non-diffuse environments," vol. 20, no. 4, 1986, ISSN: 0097-8930. DOI: 10.1145/15886.15901. [Online]. Available: https://doi.org/10.1145/15886.15901.

[2] J. T. Kajiya, "The rendering equation," vol. 20, no. 4, 1986, ISSN: 0097-8930. DOI: 10.1145/15886.15902. [Online]. Available: https://doi.org/10.1145/15886.15902.

[3] J. D. Foley and T. Whitted, *An improved illumination model for shaded display*, 1980.

[4] C. M. Goral, K. E. Torrance, D. P. Greenberg, and B. Battaile, "Modeling the interaction of light between diffuse surfaces," *SIGGRAPH Comput. Graph.*, vol. 18, no. 3, pp. 213–222, Jan. 1984, ISSN: 0097-8930. DOI: 10.1145/964965.808601. [Online]. Available: https://doi.org/10.1145/964965.808601.

[5] T. Nishita and E. Nakamae, "Continuous tone representation of three-dimensional objects taking account of shadows and interreflection," *SIGGRAPH Comput. Graph.*, vol. 19, no. 3, pp. 23–30, Jul. 1985, ISSN: 0097-8930. DOI: 10.1145/325165.325169. [Online]. Available: https://doi.org/10.1145/325165.325169.

[6] H. W. Jensen, "Global illumination using photon maps," in *Rendering Techniques '96*, X. Pueyo and P. Schröder, Eds., Vienna: Springer Vienna, 1996, pp. 21–30, ISBN: 978-3-7091-7484-5.

[7] P. Dutré, K. Bala, P. Bekaert, and P. Shirley, *Advanced Global Illumination, 2nd edition*. 2006.

[8] N. J. C. Henrik Wann Jensen, "Photon maps in bidirectional monte carlo ray tracing of complex objects," *Computers & Graphics*, vol. 19, no. 2, pp. 215–224, 1995. DOI: https://doi.org/10.1016/0097-8493(94)00145-O.