Advanced Global Illumination and Rendering

Monte Carlo Ray Tracing and Photon Mapping

Jesper Larsson (jesla966)

Julian Voith (julvo429)

25 November 2023

Linköping University

ABSTRACT

This report gives a short overview of different global illumination techniques such as Whitted ray tracing, Monte Carlo ray tracing, and photon mapping. It covers the individual strengths of each and an implementation of the Monte Carlo method. To render potential caustics, the renderer was extended with a simple photon mapping scheme that visualizes the photons shot into the scene towards a transparent sphere. The scene was rendered with multiple objects such as spheres and a tetrahedron, and different parameters varying in the number of rays per pixel, the number of shadow rays, and the resolution. In the end, parameters and their impact on speed and quality are discussed. This discussion also includes potential improvements to the renderer such as utilizing multithreading.

Contents

1 INTRODUCTION

In the realm of computer graphics, there are many ways of rendering a scene that range from simple to complex. Some methods like the Blinn-Phong illumination method use only direct illumination to approximate the lighting in the scene. The method is fast but does not result in particularly realistic lighting. Other methods that aim to produce more realistic lighting are the so-called global illumination methods. They achieve their more realistic lighting by considering not only the direct illumination but also the indirect illumination, that is they consider how light bounces around in the scene and how that contributes to the lighting situation. How they calculate the indirect illumination differs from method to method, but some examples of global illumination models are Whitted ray tracing, Monte Carlo ray tracing, and Photon mapping.

1.1 Whitted Ray Tracing

Whitted ray tracing considers only the BRDF of perfect reflections and perfect refractions. A Ray is launched from the camera into the scene. When the ray hits a surface, it is either reflected, refracted, or both depending on the characteristics of the surface. If the ray gets reflected the outgoing angle of the reflected ray is the same as the incoming ray because only perfect reflections are considered. Snell's law is used to calculate the angle of refracted rays. If the ray hits a light source, hits a non-perfect reflector or its contribution to the final color is below a threshold no more reflections or refractions are done.

The radiance is calculated by following the ray path backward towards the camera. At every intersection point, the reflected light and refracted light are combined for the radiance calculation. At every intersection, a shadow ray is also launched towards the light source to see if direct illumination should be added to the radiance calculation.

1.2 Monte Carlo Ray Tracing

Monte Carlo ray tracing considers the BRDF of all surfaces in contrast to Whitted ray tracing. This method functions similarly to Whitted ray tracing except when the ray hits a non-perfect reflector it assumes the surface is a Lambertian reflector and calculates the angle of the outgoing ray using random numbers. Then a ray hits a Lambertian reflector Russian roulette is used to determine if the ray path should continue or be terminated. More details on this method follow later in the report.

1.3 Photon Mapping

Photon mapping is a two-pass rendering method that uses photon maps to render the scene by storing the interactions of the photons in a 3D photon map. Different from Monte Carlo ray tracing photon mapping can render caustics from specular objects.

The first pass consists of emitting photons from the light source in random directions into the scene. If the photons hit an object in the scene, they can be either reflected or transmitted. When the rays interact with the scene, they are also stored in the photon maps which usually consist of a Kd-tree. Photons that come directly from the light source and hit a specular object are terminated when they hit a non-perfect reflector and are stored in the Caustics photon map. All other photons are stored in the global photon map. Shadow photons are also created by tracing photons with origin in the light source through the entire scene.

The second pass uses the photon maps to calculate the radiance in the scene. The caustics are visualized directly. All other radiance is estimated by using the surrounding photon density from the global photon map. Shadow photons are used to evaluate if shadow rays should be sent. If there are no shadow photons or a lot of shadow photons in an area no shadow rays should be sent. If there is a mix of normal photons and shadow photons a shadow ray should be sent to see if direct illumination should be considered.

2 BACKGROUND

The Implemented renderer is a Monte Carlo renderer. In this chapter, we describe the theory behind the Monte Carlo renderer and our implementation.

2.1 Bi-directional Reflectance Distribution Function (BRDF)

Each surface of an object a ray can hit has different properties. Those properties define how the ray behaves on intersection with this surface. For instance, a ray can be scattered or reflected depending on the surface it hits. To create photorealistic images, surface properties need to be considered. These properties are described mathematically in a linear *bi-directional reflectance distribution function*, or short BRDF, as can be seen in equation 1.

$$f(x, \theta_i, \theta_o) \tag{1}$$

where f is the fraction of light at point x reflected in the outgoing direction $\theta_o$given the light coming from the incoming direction θ_i

This general equation can also be found in the rendering equation 9**.** Different types of BRDFs have been developed, which can be used for portraying a variety of realistic surface behaviors within rendering.

2.2 Lambertian Model

The *Lambertian reflection model* is used to represent general matte surfaces. The rays hitting a Lambertian surface, are being reflected diffusely. This means all rays hitting point x will be reflected in all directions. This simple model uses a constant function for all directions which is given by equation 2.

$$f(x, \theta_i, \theta_o) = \frac{\rho}{\pi} \tag{2}$$

Where ρ is a reflection coefficient which is normalized by π. A Lambertian reflector has the same values for any combination of θ_i and θ_o, hence we can omit these parameters in the equation.

2.3 Specular Reflection

The BRDF for specular reflection is rather simplified. The model assumes that the specular surfaces behave like perfect mirrors. Perfect mirrors reflect the ray in the same angle and radiance as it arrives at point x. The outgoing reflected ray can be calculated as in equation 3.

$$d_o = d_i - 2(d_i \cdot N)N \tag{3}$$

Where d_i is the direction of the ray which is emitted from the eye and N the normal of the surface the ray intersects with.

2.4 Reflection and refraction on transparent object

Transparent objects are handled in a different way when rendering. Once a Ray hits a transparent object, it is reflected and refracted. Hence, the ray is split up into 2 new rays. One which is being reflected as described in chapter 2.3 and the other enters the transparent object. In our implementation, the ray which is hitting the surface is not split up. Upon intersection, we randomly decide if the ray is being reflected or refracted.

For a refracted ray, the refraction angle β needs to be calculated. To calculate β**,** Snell's law can be used as described in equation 4,

$$\frac{sin\Omega}{sin\beta} = \frac{n_2}{n_1} = R^{-1} \tag{4}$$

where Ω is the angle between the incoming reflected ray and the surface normal, β is the angle between the incoming refracted ray and the surface normal, n_1 and n_2 are the refraction indices for air and glass and transition value R is the transition value from air to glass.

With Snell's law, we can calculate the direction of the refracted ray. This is given by equation 5,

$$d_{refr} = Rd_o + N\left(-R(N \cdot d_o) - \sqrt{1 - R^2(1 - (N \cdot d_o)^2)}\right) \tag{5}$$

Where R is the transition value from air to glass, d_o the direction of the outgoing ray (to the eye), and N is the normal of the point X.

The transition value R is calculated by using Snell's law in equation 6.

$$R = \frac{n_1}{n_2} \tag{6}$$

Where n_1 and n_2 are the refraction indices for air and glass. The ratio going from air to glass is as in equation 6. The ratio from glass to air is calculated as stated in Snell's law in equation 4.

The split of the rays also changes the BRDF. The correct BRDF splits the incoming radiance between the refracted and reflected rays. The correct radiance can be computed with Fresnel's formula. Unfortunately, this formula is quite computationally heavy. However, we can approximate it by using Schlick's formula which is given by equations 7 and 8.

$$R_0 = \left(\frac{n_1 - n_2}{n_1 + n_2}\right) \tag{7}$$

$$R(\Omega) = R_0 + (1 - R_0)(1 - \cos\Omega)^5 \tag{8}$$

Where Ω is the angle of the outgoing (incoming) radiance (importance) ray. n_1 and n_2 are the described refraction indices of the two media. The radiance of the reflected ray is represented by $R(\Omega)$ and the radiance of the refracted ray is given by equation 9.

$$T(\Omega) = 1 - R(\Omega) \tag{9}$$

where $T(\Omega)$ is the radiance of the refracted ray,

These calculations are the same both ways, air to glass and glass to air.

Additionally, if the ray is inside the transparent object we also must check for total internal reflection, that is if the ray is entirely reflected inside the sphere without having any chance of being transmitted outside. We use equation 10 for this.

$$\left(\frac{n_1 \sin\Omega}{n_2}\right) < 1 \tag{10}$$

Where n1, n2 are the refraction indices and $sin\Omega$ is gotten from the incoming ray. If equation 10 is smaller than 1 we have both a reflected ray and a refracted ray and they are handled as described above. If equation 10 is greater than one, we have total internal reflection and only a reflected ray is needed where all the incoming light is reflected.

2.5 Monte Carlo Ray Tracing

In Monte Carlo ray tracing, rays are sent out from the camera into the scene similarly to Whitted ray tracing. Those rays are carrying *importance*, which behaves identically to radiance but goes in the opposite direction. In the real world, when rays of light are intersecting with a diffuse surface, the light is scattered in an infinite amount of hemispherical bounded directions. The Monte Carlo method approximates how these infinitely reflected rays of light are spread and captured by the camera. It approximates it because it is not possible to integrate over an infinite number of incoming directions. The Monte Carlo algorithm approximates the radiance at a point x in the direction $\theta_o$as follows in equation 11.

$$L(x \rightarrow \theta_o) = \int_\Omega f_r(x, \theta_i, \theta_o) L(x \leftarrow \theta_i) \approx \frac{\pi}{N} \sum_{i=1}^{N} f_r(x, \theta_i, \theta_o) L(x \leftarrow \theta_i) \tag{11}$$

Where N is the number of generated Rays per pixel. The rays which are being sent out, follow the same set of rules in terms of reflection and refractions discussed in chapter 2.4. Additionally, they terminate once they hit a light source.

As discussed earlier, once a ray hits a diffuse surface, a bordering on an infinite number of rays is being produced. Hence, we need a mechanism to terminate those generated rays. This mechanism, however, needs to be unbiased. We are using a termination scheme called *Russian roulette*.

2.6 Russian Roulette

Traditionally, Whitted Raytracing always cut off at Lambertian reflectors. This does not work with Monte Carlo ray tracing. To enable photorealism, we want to use a photorealistic scheme, this is scheme is called Russian roulette. As the name suggests, the scheme is based on probabilistic outcomes. The probability that a ray is cut-off is $0 \leq p_k \leq 1$. Hence, the probability that the ray survives is $p_s = 1 - p_k$. But simply using the chance of the ray surviving or being terminated in the radiance estimation means that the amount of radiance is underestimated. The solution to this is to set the probability that the ray survives to be equal to the reflection coefficient ρ, the same value used to calculate the BRDF in equation 2. This solves the problem of underestimating the radiance and gives a photorealistic result.

If a ray survives, a new ray is sent in a random direction from the previous ray's intersection with the surface. The random direction the new ray follows is calculated using equation 12.

$$\begin{cases} \varphi_i = 2\pi y_i \\ \Omega_i = \cos^{-1}\sqrt{1 - y_i} \end{cases} \tag{12}$$

Where the φ_i is the azimuth, Ω_i is the inclination of the new ray, and y_i is a random number between zero and one.

2.7 Direct Light Contribution

Light sources are only a small part of the scene. Most of the rays won't ever reach the light source, which causes the issue that these rays will not contribute to the image since the light is the emitting object in a scene.

To make this more efficient and to render images faster, we can take light directly along random points into account. This is done by testing each intersection point with the light source. To do so, we are sending shadow rays for each intersection point to the random points on the light source. The direct light contribution from one light at a point x in the direction θ_o can then be estimated by equation 13.

$$\langle L_D(x \to \theta_o) \rangle = \frac{AL_e}{N} \sum_{i=1}^{N} f(x, d_i, \theta_o) V(x, y_i) \frac{\cos\Omega_{x,i} \cos\Omega_{y,i}}{\|d_i\|^2} \tag{13}$$

where A is the area of the light source, L_e is the radiance leaving the light at any point in any direction. $\cos\Omega_{x,i}$ can be obtained by $N_x \cdot d_i/||d_i||$ and $\cos\Omega_{y,i}$ by $N_y \cdot d_i/||d_i||$ where N is the corresponding y or x value of the normal of the light point and d_i the direction of the shadow ray which is arriving at the light point. $\|d_i\|^2$ is the distance between the shadow ray and the light point. $V(x, y_i)$ represents the visibility term, which is 1 if the shadow ray is not blocked by another object and 0 when it is blocked.

2.8 Photon Mapping

Photon Mapping is a rendering technique developed by W. Jensen in 1995 [1] The goal of photon mapping is to render a scene with global illumination more efficiently, by approximating the rendering equations, using a two-pass method.

The first pass turns around the perspective with traditional Monte Carlo ray tracing. It simulates the light propagation in a scene from the light source's perspective. The light source is emitting a huge number of photons, carrying small amounts of flux, into the scene. These photons/rays are hemispherically bounded and shot into random directions from random points on the light source's surface. Each generated ray uses a path tracing algorithm that supports realistic reflection and refraction on specular and diffuse surfaces. The algorithm works equally as described in the previous chapters. Eventually, each ray terminates on a diffuse surface. The point where the ray is terminated along with its incoming direction and flux is stored in the photon map. However, this is also done with each bounce of the ray. The photon map itself is a data structure called k-d tree.

Additionally, during this first pass, shadow photons are created. Shadow photons are created by following the photons from the light source until their first intersection with an object. We then follow through this intersection point until we hit the other side of the object and create a shadow photon at this position and store it in the photon map.

During the second pass, the rendering pass, information is collected from the generated photon map by the ray tracer. The ray tracer estimates the radiance leaving a point x in the direction θ_o as seen in equation 14.

$$L(x \to \theta_o) = \sum_{I=1}^{N} f(x, \theta_i, \theta_o) \frac{\phi_i}{\pi r^2} \qquad (14)$$

Where θ_o is the direction of the outgoing ray, θ_i the direction stored in the photon, f represents the BRDF, $\phi_i$the flux arriving at x from the direction θ_i, and πr^2 the search range over which the radiance is collected.

During this pass, we also use the shadow photons to estimate shadows. If there are a lot of shadow photons in the search range, we assume the location is in shadow, if there are few shadow photons, we assume the location is in direct light and if there is a mix of shadow photons and light photons we send a shadow ray to check for direct light.

Compared to the technique described in W. Jensen's paper, our renderer is using a simplified version. Only one photon map for caustics is generated and used. Further, rays are only being shot toward transparent objects. Due to time constraints, a full implementation of the photon map renderer was not done, instead, the photon map was created and visualized directly to show the effects of caustics.

2.9 Implementation

The renderer was implemented from scratch using C++, OpenGL Mathematics [2]. To create the rendered image, the EasyBMP [3] library is used.

The photon map was implemented by using a k-d tree. Therefore, we used a library called libkdtree++ [4].

2.10 Camera

The camera consists of an eye point and the camera plane. These two components define the direction of the rays. The camera plane is subdivided into pixels. Using a Monte Carlo Ray Tracer, the color is computed by shooting multiple rays per pixel in random directions into the scene. Each ray is obeying the rules defined in the previous chapters. Each ray is then contributing a fraction of the final color.

2.11 The Scene

A scene was created that contained a diffuse hexagonal room consisting of 10 triangles, a diffuse tetrahedron consisting of 5 triangles, a mirror sphere, and a transparent sphere. To render this scene the intersection points between the rays and the objects needed to be calculated. This was done differently for triangles and spheres.

2.12 Intersection with triangles

To calculate the intersection, point for triangles, the Möller Trumbore algorithm was used. The intersection point is given by equation 15.

$$x(t) = p_s + t(p_e - p_s) \tag{15}$$

Where x(t) is the intersection point with the triangle, p_s is the start point of the ray, $(p_e - p_s)$ is the direction of the ray, and t is how far along the ray we travel until we hit the triangle. We calculate t using equation 16,

$$\begin{pmatrix} t \\ u \\ v \end{pmatrix} = \frac{1}{P \cdot E_1} \begin{pmatrix} Q \cdot E_1 \\ P \cdot T \\ Q \cdot D \end{pmatrix} \tag{16}$$

Where t is how far along the ray we travel until we hit the triangle, u and v are the barycentric coordinates of the equation of the triangle, and Q, E_1, P, T, D are calculated in equation 17.

$$\begin{cases} T = p_s - v_0 \\ E_1 = v_1 - v_0 \\ E_2 = v_2 - v_0 \\ D = p_e - p_s \\ P = D \times E_2 \\ Q = T \times E_1 \end{cases} \tag{17}$$

Where p_s is the ray start point, v_0, v_1, v_2 are the corner points of the triangle, and $p_e - p_s$ is the ray direction.

We use the barycentric coordinates u and v calculated in equation 16, to determine if we have intersected with the triangle by seeing if they fulfill equation 18.

$$\begin{cases} u \geq 0 \\ v \geq 0 \\ u + v \leq 1 \end{cases} \tag{18}$$

If the calculated barycentric coordinates fulfill equation 18, the ray intersects with the triangle, otherwise, it does not intersect. We then check if t > 0 because otherwise, the intersection point happens behind the ray start point. If t> 0 and the barycentric coordinates are correct, the intersection point is given as in equation 16.

2.13 Intersection with spheres

The intersection with the sphere is calculated using equation 19.

$$x_r(t) = S + Dt \tag{19}$$

Where $x_r(t)$ is the intersection point, S is the ray start point, D is the ray direction and t is how far along the ray direction we need to travel to hit the intersection point.

T is calculated using equation 20.

$$t = \frac{-c_2 \pm \sqrt{c_2^2 - 4c_1c_3}}{2c_1} \tag{20}$$

Where t is how far along the ray direction, we need to travel to hit the intersection point, and c_1, c_2, c_3 are calculated using equation 21.

$$\begin{cases} c_1 = D^2 \\ c_2 = 2D(S - C) \\ c_3 = (S - C)^2 - r^2 \end{cases} \tag{21}$$

Where D is the ray direction, S is the ray start point, C is the center point of the sphere and r is the sphere radius.

Depending on the value under the square root, equation 20 has a different number of solutions. If the square root contains less than 0 the ray misses. If it is equal to zero, the ray touches the sphere only once. If it is greater than zero there are 2 solutions, meaning two intersection points. If equation 20 gives two different values for t choose the smaller t to calculate the intersection point using equation 19.

3 RESULTS

Figures 1 – 5 feature a scene rendered with the implemented Monte Carlo renderer with different values for its parameters. The scene consists of a room with diffuse walls with a light source in the ceiling, a diffuse tetrahedron, a mirror ball, and a transparent sphere. All parameters used for the different renders and the time for each render are collected in Table 1. The right image in all the figures is the same and is used for comparison. In the table, it is called the Comparison image. Figure 6 is a visualization of the photon map of a scene rendered with our photon mapper. The scene is the same as it was for our Monte Carlo renderer except the mirror sphere has been replaced by a transparent sphere. All scenes were rendered on a computer with an AMD Ryzen 7 6800H CPU and 32 GB of RAM.

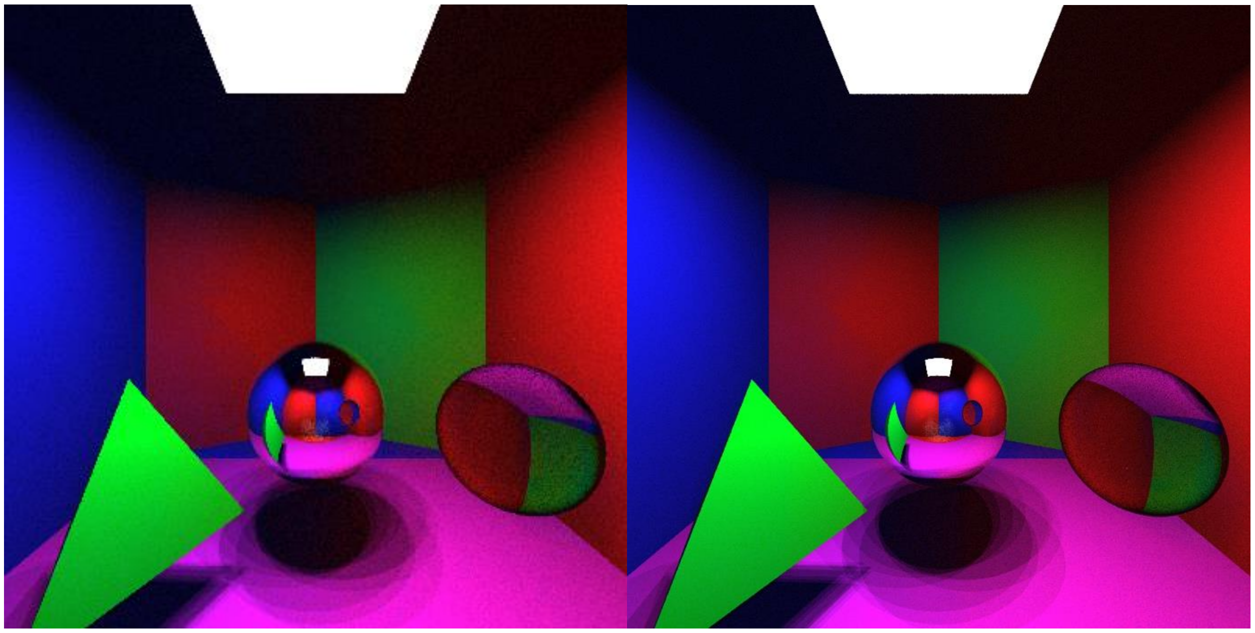
Figure 1: The scene rendered with different resolutions. The left scene was rendered in 400 by 400-pixel resolution and the right scene in 800 by 800-pixel resolution.

As can be seen in Figure 1, an increase in the resolution reduces the effect of noise in the scene without affecting the colors of the objects in the scene. This means changing the resolution of the resulting image only changes its quality without changing its content.

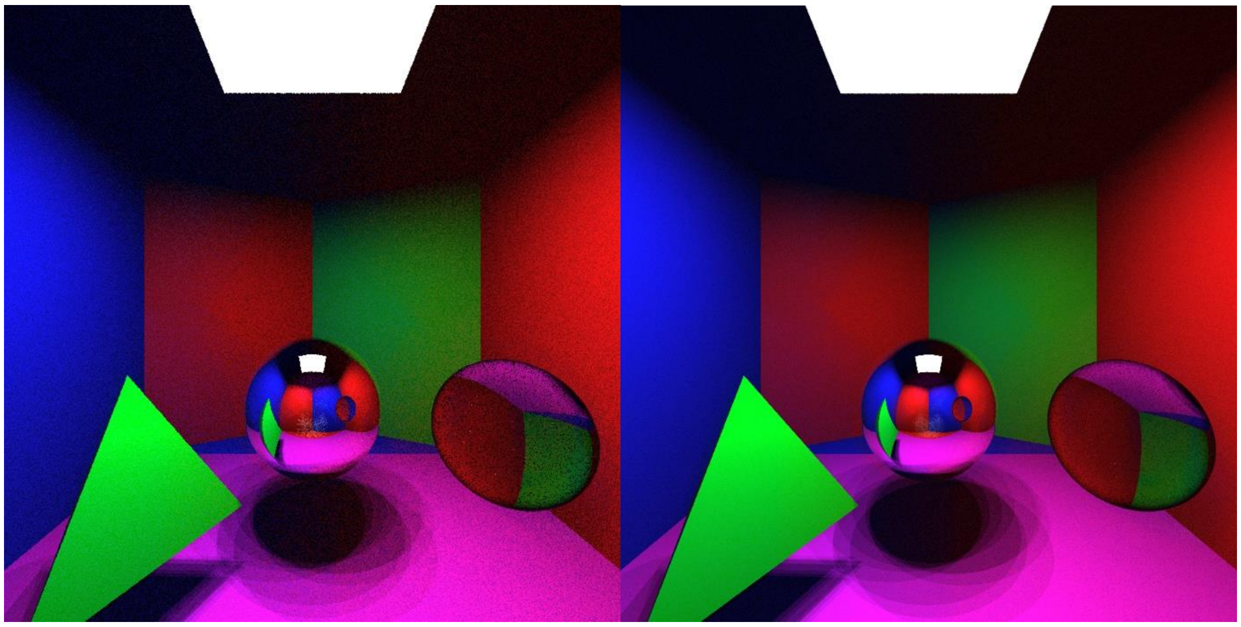

Figure 2: The scene rendered with different amounts of rays per pixel. The left scene was rendered with 1 ray per pixel and the right with 5 rays per pixel.

As can be seen in the left image of Figure 2, the amount of noise in the scene is quite high with only 1 ray per pixel but when we increase the number of rays per pixel the noise in the scene reduces drastically as can be seen in the right image.

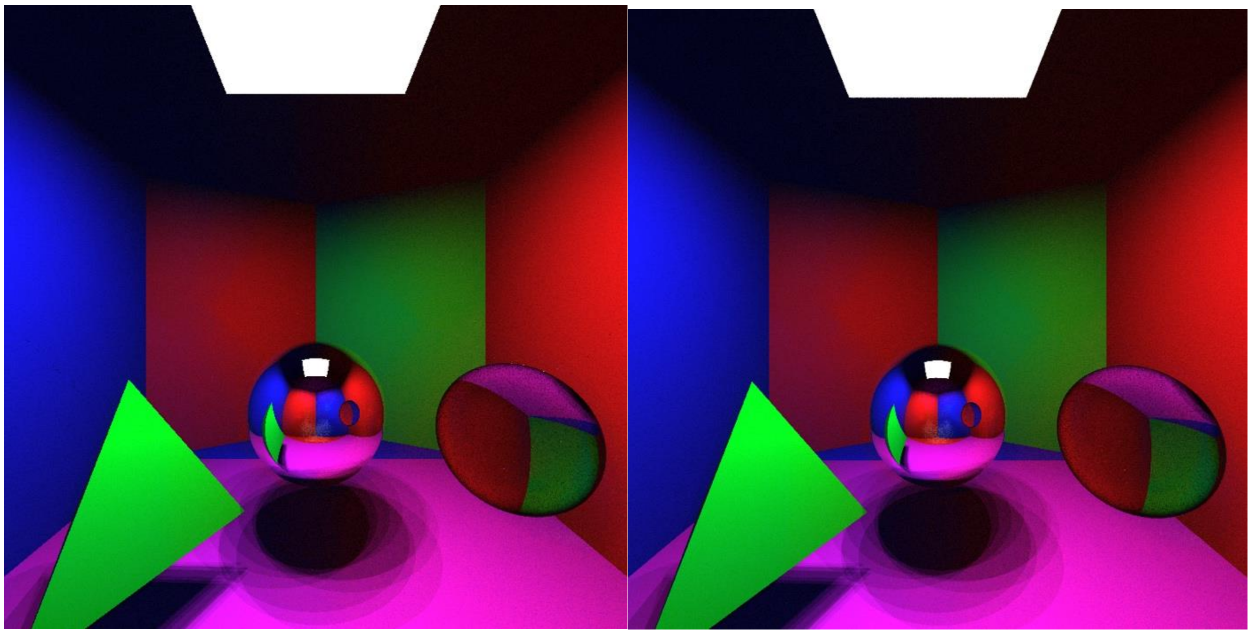

Figure 3: The scene rendered with and without noise in the ray direction from the eye. The left scene does not have noise while the right scene does. Both scenes were rendered with 5 rays per pixel.

While not as obvious as in Figure 2, a close inspection of Figure 3 reveals that introducing noise in the ray direction from the eye reduces the amount of noise in the scene.

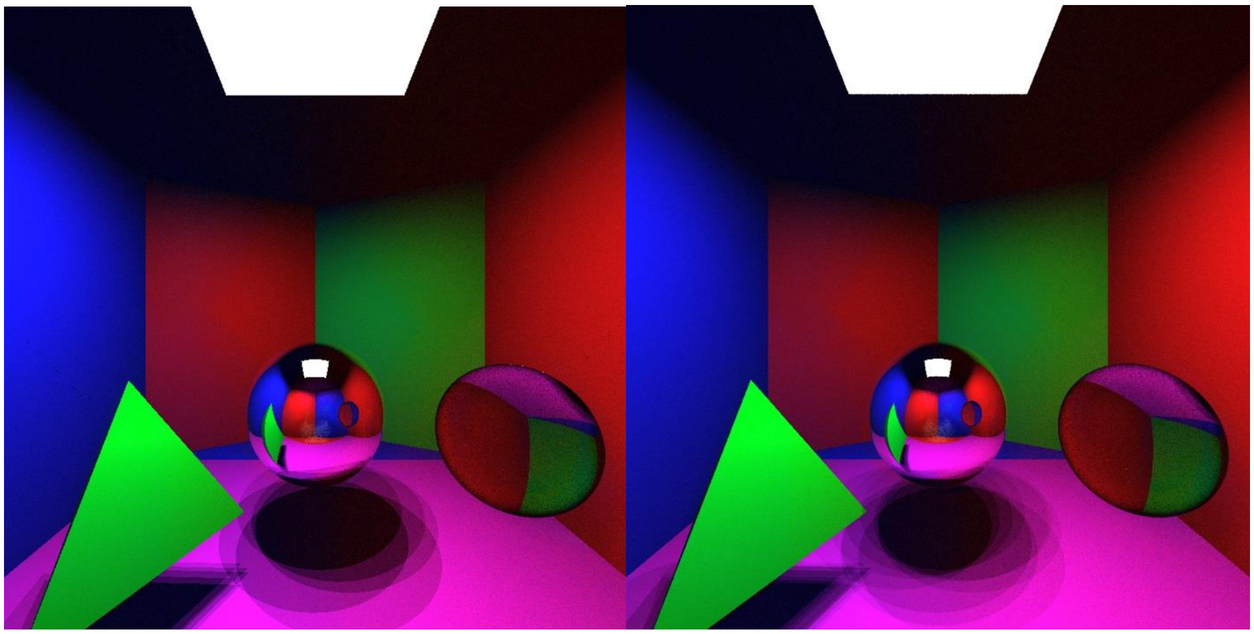

Figure 4: The scene rendered with different amounts of shadow rays. The left scene was rendered with 5 shadow rays and the right with 10 shadow rays.

Figure 4 shows that the amount of shadow rays directly correlates to the softness of the shadows. An increase in shadow rays gives a softer shadow while a decrease in shadow rays gives a harder shadow.

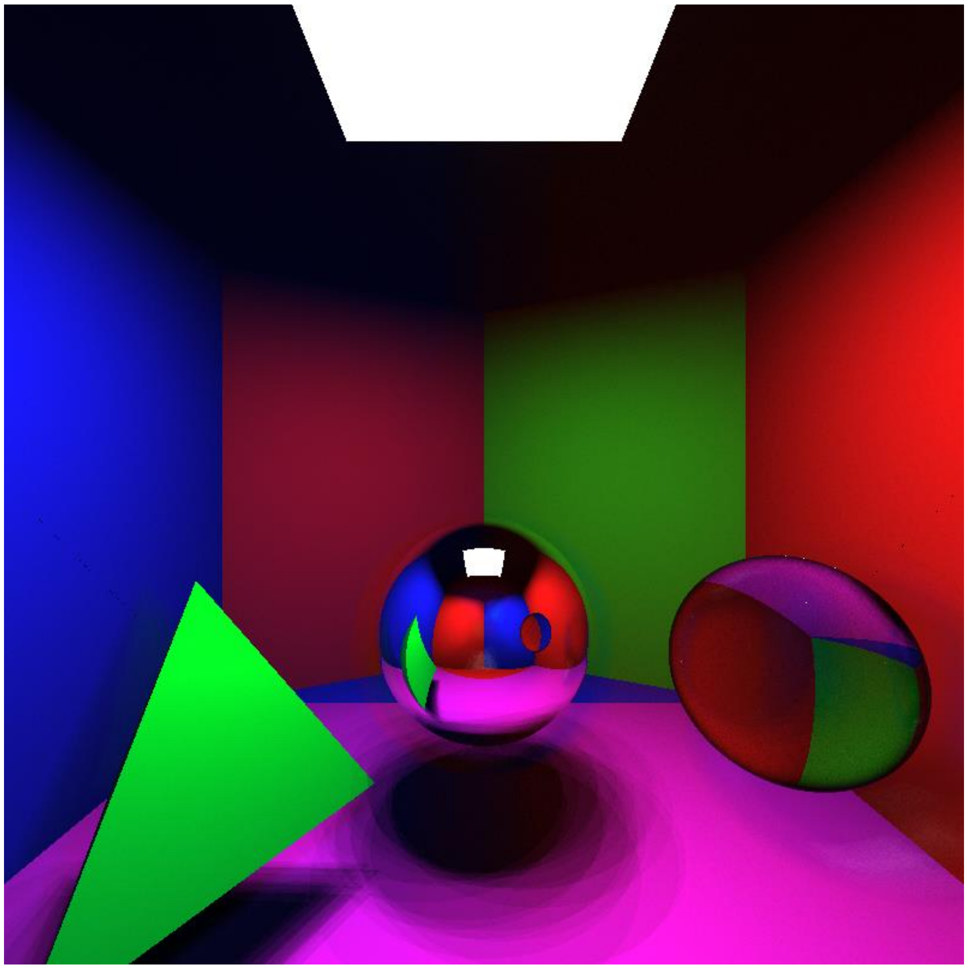

Figure 5: Shows the final rendering of this project. The scene was rendered using 30 rays per pixel and 20 shadow rays.

In Figure 5 you can see the quality increase that comes with an increase in rays per pixel. The quality has increased to such an extent that you can make out the faint reflection of the mirror sphere in the transparent sphere. The color is consistent across the surfaces with minimal noise. You can also see the effect the transparent sphere has on the light on the right wall.

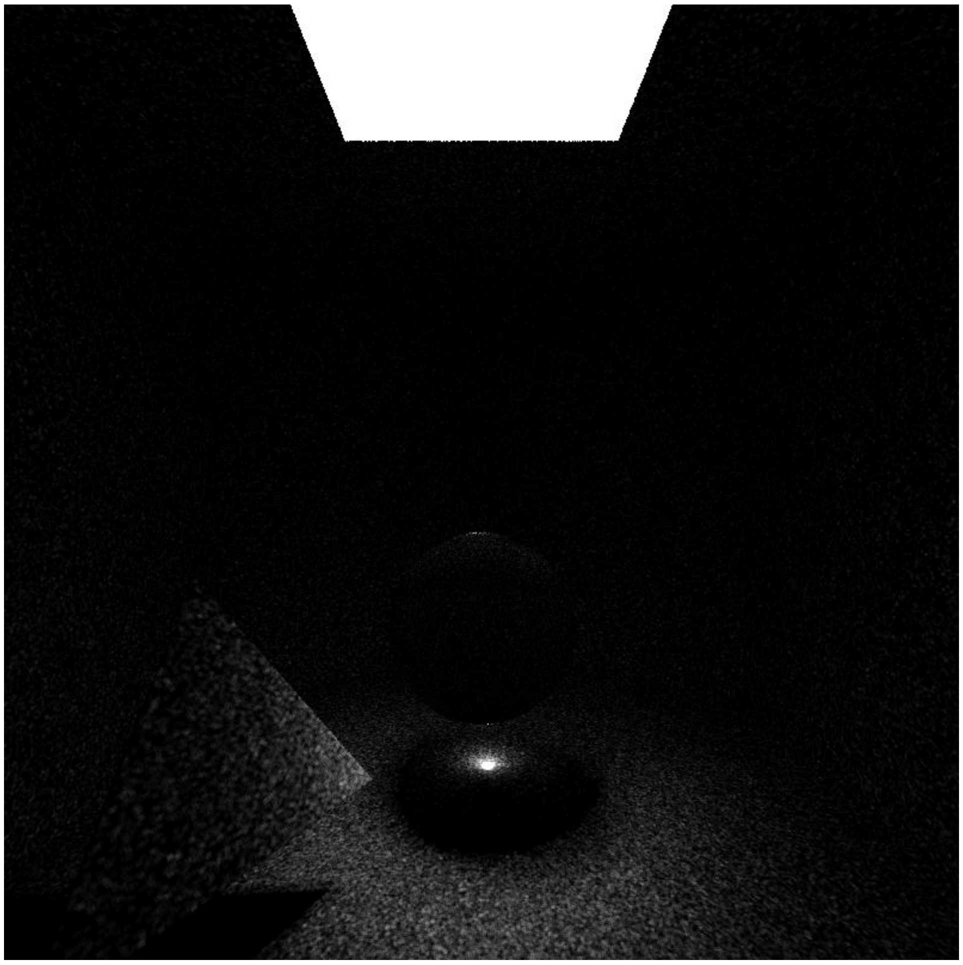

Figure 6: Shows the direct visualization of the photon map. The photon map was created with 3 light points and 1 million photons per light point.

Figure 6 clearly shows the effect of caustics created by the transparent sphere. The light that has entered the transparent sphere has been focused on a single spot under the sphere. This means that the area around this spot is much darker as almost no photons reached these areas. The creation of the photon map and the rendering took 96.63 seconds.

Table 1 Comparison table

Image	Resolution	Rays per pixel	Shadow rays	Ray noise	Time (seconds)
Comparison image	800 by 800	5	10	yes	115.46
Figure 1	400 by 400	5	10	yes	29.15
Figure 2	800 by 800	1	10	yes	23.64
Figure 3	800 by 800	5	10	no	117.68
Figure 4	800 by 800	5	5	yes	88.93
Figure 5	800 by 800	30	20	yes	1 601.34 (≈27 minutes)

4 DISCUSSION

4.1 Monte Carlo Ray Tracing

Table 1 shows that the most time-consuming parts of the Monte Carlo rendering equation are the resolution, number of rays per pixel, and amount of shadow rays. Adding noise to the rays from the eye does not affect the time much but reduces the amount of noise in the scene, meaning it should be added to the render if possible. Adding more rays per pixel and more shadow rays gives a compounding effect on the amount of time required as every ray created sends shadow rays.

There are a few ways to reduce the amount of time it takes to render an image. One way is to reduce the number of rays per pixel and shadow rays but that also reduces the quality of the resulting image. If you are rendering a scene where the shadows are less important, reducing the number of shadow rays can be a good option. Another way to reduce the time without affecting the quality is to use fewer rays per pixel but increase the resolution of the render to be higher than the desired resolution. By later scaling down the resulting picture the effect of any noise introduced by the reduced ray count can be hidden. Another way is to speed up the renderer itself. The implementation in this paper renders only one pixel at a time. The way the Monte Carlo rendering technique works means that the color of each pixel is computed completely independently from the other pixels, meaning it is well suited for multithreading, that is using a computer with multiple computing cores to calculate the color of multiple pixels at the same time. This would speed up the rendering process significantly without needing to reduce the quality. This was not done during this project due to a lack of time.

4.2 Photon Mapping

As a full photon map renderer was not implemented, we cannot compare the results between the photon mapper and our Monte Carlo renderer directly on quality, but some differences can still be discussed. If we compare Figure 6 to Figure 5, the effect of caustics is not noticeable in the Monte Carlo rendering. This shows that photon mapping is very good at rendering caustics compared to Monte Carlo. While it is not fully fair to compare the speed of the renders as our photon mapper is not complete, generally photon mappers are faster at rendering a scene than Monte Carlo renderers. One of the reasons for this is the use of shadow rays in Monte Carlo. To get shadows in Monte Carlo ray tracing, shadow rays need to be shot for every ray at every intersection point. This is not needed in a complete photon map renderer. A photon map renderer uses shadow photons which dramatically reduce the amount of shadow rays needed in a scene without affecting the render quality. As seen above, shadow rays are one of the main contributors to an increase in rendering time. Another good attribute of the photon mapping technique is the fact that it is camera independent. This means that the photon map can be created and then used to render the scene using multiple cameras from different positions. Monte Carlo on the other hand needs to render the scene fully every time the camera is changed.

4.3 Further Improvements

There are a few ways to go from here to improve the quality of the renders.

For the Monte Calo renders, one way is to simply increase the number of rays per pixel and shadow rays. This would increase the quality at the expense of time. Another way is to use a different rendering method such as photon mapping. As stated above photon mapping has a few advantages over Monte Carlo, such as a generally lower rendering time and the ability to render caustics.

There are also a few ways to increase the quality of a photon mapper. The simplest way is to increase the amount of photons sent from the light source, as the cost of time and memory. Photon mapping can also be implemented so that it considers participating media. Participating media can be smoke, dust, or cloudy liquid. That means that photon mapping enables a more general rendering technique that can be used in more complex scenes and gives a more accurate result.

Bibliography

[1] H. W. Jensen, "Global Illumination using Photon Maps," *"Rendering Techniques '96". Eds. X. Pueyo and P. Schröder.,* pp. 21-30, 1996.

[2] "OpenGL Mathematics," [Online]. Available: https://glm.g-truc.net/0.9.9/. [Accessed 29 10 2023].

[3] P. Macklin, "EasyBMP," [Online]. Available: https://easybmp.sourceforge.net/. [Accessed 29 10 2023].

[4] "libkdtree++," [Online]. Available: https://github.com/nvmd/libkdtree. [Accessed 29 10 2023].