A Monte Carlo Ray Tracer

Fredrik Johansson, Isak Engström

TNCG15 - Advanced Global Illumination and Rendering
Department of Science and Technology, Linköping University

November 7, 2019

Abstract

This report is based on the implementation of a Monte Carlo Ray tracer made during the course Advanced Global Illumination and Rendering at Linköpings University. The ray tracer is physically based and were implemented in C++. The underlying physics and mathematics are presented in the background section, ranging from Bidirectional Reflectance Distribution Functions to Estimations of the light contribution from the rendering equation. In the result, different images produced by the ray tracer are presented. These images have different set parameters to illustrate certain features of the ray tracer. The result is analysed and discussed in the last section.

1 Introduction

Physically based rendering (PBR) is an approach in computer graphics that seeks to render photorealistic images by accurately modelling the flow of light in the real world. The goal is to create a *Globally Illuminated* scene which means that indirect light contribution has to be evaluated.

In 1980 Turner Whitted introduced *Whitted ray tracing* [1], which is an example of a method that does not take indirect light contribution into account. Whitted ray tracing exhibit the idea of using *radiance* and *importance*. Importance flow from the eye into the scene while the radiance flow in the opposite direction. At every intersection a ray is emitted towards the light source called a *shadow ray*. This ray checks if the intersection is behind an object and therefore in that objects shadow.

Since most of the rays are not going to reach the light source the rays would get zero light contribution. To counter this, Whitted ray tracing has to have a stopping condition. Common stopping conditions are:

- Stop if the ray hits a diffuse reflector or a light source.
- Stop when the importance of a ray gets below a certain threshold.
- Stop when the depth of a ray reaches a certain number.

The radiance of each intersection will transfer up the ray paths and add radiance according to the importance at each intersection. This method will make the shadows sharp with high contrast. This can be countered by adding ambient lighting which is not very realistic.

An example of a global illumination model which was introduced in 1950 and adapted for computer graphics in 1984 is the *Radiosity method*. Instead of using rays, this method incorporates surface elements known as *patches*. The method received its name because of the use of *radiosity*, a radiometric unit, defined as the outgoing *flux* over a surface area. Flux indicate how much energy is absorbed or emitted by a surface element over a certain time interval. It is hence not an attribute of any ray. The scene consists of patches and it is assumed to be made up by *Lambertian reflectors*. Lambertian reflectors describe purely matte, or diffusely reflective surfaces. It is a global illumination model, as the radiosity traverse from one patch to another, depending on the visibility between each pair. This leads to illumination contribution in-between patches and not only directly from the light source.

The radiosity method gives physically accurate results, however, its use is restricted. Partly because of the scene is restrained to Lambertian reflectors, but also because it is difficult to combine with tessellation algorithms [2].

A common model for global illumination is the well known *rendering equation* which is presented in (1).

$$L_s(x, \Psi_r) = L_e(x, \Psi_r) + \int_\Omega f_r(x, \Psi_i, \Psi_r) L_i(x, \Psi_i) cos\theta_i d\omega_i \tag{1}$$

This equation was introduced by James Kajiya in 1986 [3]. The equation integrates over an infinite number of incoming rays, therefore, approximations have been made to integrate over a finite number of rays instead.

The radiosity method has since 1986 been adapted by simplifying this rendering equation. The patches are approximated to be triangles. The radiosity transferred from a triangle to another is computed as a product between the constant radiosity of the emitting triangle and the so-called *form factor*, which contain all the geometric dependencies. Further simplifications result in a linear equation for each pair of triangles. Combining this fact with the constant *Bidirectional Reflectance Distribution Function* (BRDF) of Lambertian surfaces, these calculations can be performed off-line for static scenes, independent of viewing direction. The radiosity scheme can therefore be combined with ray tracing by calculating the *view independent* and *view dependent* parts separately. By baking the diffuse light beforehand, the radiosity method could also be combined with real-time rendering [4].

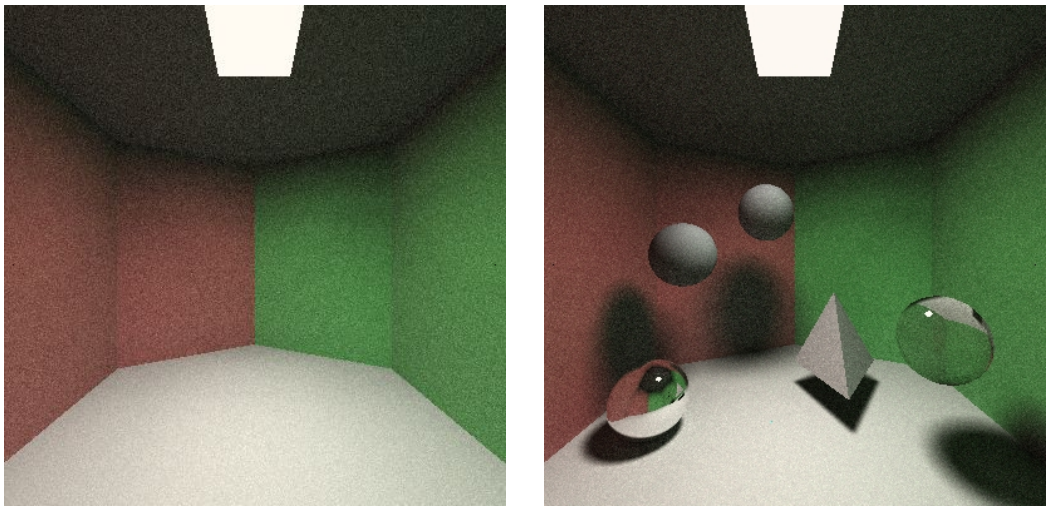
One unbiased method to approximate the rendering equation is called *Monte Carlo*. When firstly introduced, the Monte Carlo method led to what seemed to be unavoidably noisy images. The reason was the variance created during integration. However, with the application of the rendering equation introduced in Kajiya's paper, the noise could be reduced. Monte Carlo picks a random direction using a uniform random number generator. The integral is evaluated using this direction. This technique makes it possible to calculate indirect light contribution. This will give the image smoother shadows and it will also introduce *color bleeding*. A further description of color bleeding is found in section 2.5.4.

For a Monte Carlo ray tracer new stopping conditions have to be introduced. One alternative is to give the diffuse surfaces an *absorption coefficient*. However, this will give the model a biased result. This can be countered by introducing *Russian Roulette*. Russian roulette is a method which terminates the ray in an unbiased manner [5].

This paper will feature an implementation of a Monte Carlo ray tracer with the aim of rendering a photorealistic image. The scene which is used is not the traditional Cornell Box introduced in 1984 [6]. Instead the scene consists of six walls in a hexagonal shape. The camera is positioned along a line between two opposite corners facing the same direction as the line. The walls, ceiling and floor are all Lambertian. Furthermore, the three walls to the left are red whilst the right ones are green. Both the ceiling and the floor are colored white. Lastly, there is an area light source in the center of the ceiling

in the shape of a square. The empty scene is shown in Figure 1a.

Figure 1b display a scene with different objects with different materials or reflectors. There is a tetrahedron and four spheres in the scene. The tetrahedron has a Lambertian material. The spheres have, ordered from left to right, *pure reflective*, *Oren Nayar*, Lambertian and *Transparent* properties. These are described in further detail in section 2.3. The spheres are defined as *iso-surfaces* whilst the other objects are defined by triangles.



(a) An empty scene

(b) A scene with object of different shapes and materials

Figure 1: The scene used for benchmarking

2 Background

The Monte Carlo scheme for estimating integrals was used for this implementation. The implementation was written in C++ using the library GLM. A tree data structure was used to store ray intersections and the render loop was optimized using the multi-processing application programming interface OpenMP.

2.1 Intersections

Intersections are stored in a tree structure where every intersection contains information about the properties of the intersected object, such as material and color. The intersections are computed using a *Möller Trumbore algorithm*.

A ray is emitted from a specific point in the scene towards the center of every pixel. For every emitted ray an instance of a tree is initialized. The emitted ray will reflect and refract dependant on the BRDF of the intersected surface. Once a ray is terminated its light contribution can be calculated by evaluating its direct light contribution. The total light contribution is then recursively calculated by post-traversing the tree while adding the light contributions from each intersection. Eventually the light contribution for the root intersection is calculated which equals the total light contribution for the emitted ray.

2.2 Monte Carlo Integration

Consider the integral seen in (2) with a function $f(x)$ such that the integral is unsolvable. For simplicity, the example is illustrated in one dimension. However, the concept is true for higher dimensions as well. The integral can be estimated using the Monte Carlo method. The estimator $\langle I \rangle$ of the given unsolvable integral I can be seen in (3).

$$I = \int_a^b f(x)dx = E[g(x)] \tag{2}$$

$$\langle I \rangle = \frac{1}{N} \sum_{i=1}^{N} g(x_i) = \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{p(x_i)} \tag{3}$$

Where E is the estimated integral value, N is the number of samples and p is a uniform distribution function. N samples are drawn for x_i using $p(x)$. Then $f(x)$ and $p(x)$ are sampled at the values x_i. This will yield an estimate that will converge to the expected value E when N goes toward infinity.

2.3 Bidirectional Reflectance Distribution Functions

The Bidirectional Reflectance Distribution Function is a scalar function of multiple dimensions that determines how different material reflect light. In this implementation three dimensions were used to simplify the different wavelengths of light into a single value. The BRDF formula can be seen in (4).

$$f_r(x, \Psi_i, \Psi_r) \tag{4}$$

Where x is the wavelength of the light, Ψ_i is the incoming ray and Ψ_r is the reflected ray. In this project four different materials were used:

- Diffuse Lamertian reflectors
- Diffuse Oren-Nayar reflectors
- Pure reflective surfaces
- Transparent surfaces

The Lambertian reflectors emits rays in all directions with the magnitude determined by the BRDF according to (5).

$$f_r = \frac{\rho}{\pi} \tag{5}$$

Where ρ is a constant reflection coefficient with values $\epsilon[0, 1]$.

Oren-Nayar is a further developed version of the Lambertian reflector where a specified value for the roughness is set, which corresponds to the standard deviation of the *Gaussian*. This value determines how the light scatter. Help variables A and B are computed with the roughness value, seen in (6).

$$A = 1 - \frac{\sigma^2}{2(\sigma^2 + 0.33)}, \quad B = \frac{0.45\sigma^2}{\sigma^2 + 0.09} \tag{6}$$

Where σ is the standard deviation of the Guassian. With A and B the final Oren-Nayar BRDF is computed with (7).

$$f_r = \frac{\rho}{\pi}(A + B \max(0, \cos[\varphi_{in} - \varphi_{out}]) \sin\alpha \sin\beta) \tag{7}$$

Where φ_{in} and φ_{out} are the azimuth angles, α is the largest value of the in- and out going inclination angles and β is the smallest value of the in- and out going inclination angles.

By evaluating (7) it is possible to conclude that a Oren-Nayar surface with roughness set to zero is equal to a Lambertian reflector, since B would become zero.

The pure reflective surfaces reflects all light in a direction dependant on the incoming rays absolute angle. The transparent surfaces split the ray into a reflected ray and a transmitted ray. These rays are computed with Schlick's approximation formula, which is an approximation of Fresnel's equation. First a constant is computed using (8). Then the reflective part is computed with (9) and the refracted part with (10).

$$R_0 = \left(\frac{n_1 - n_2}{n_1 + n_2}\right)^2 \tag{8}$$

Where n_1 is the refracting index of the current medium and n_2 is the retracting index of the intersected material.

$$R(\theta) = R_0 + (1 - R_0)(1 - cos\theta)^5 \tag{9}$$

Where $R(\theta)$ is the multiplier for the reflected ray and θ is the angle between the incoming ray and the surface normal.

$$T = (1 - R(\theta)) \tag{10}$$

Where T is the multiplier for the refracted ray. Care has to be taken when the incoming ray is coming from a medium with a higher refracting index than the intersected materials index. This is done by evaluating the *Brewster angle* according to (11).

$$\alpha_m = \arcsin \frac{n_2}{n_1} \tag{11}$$

Where α_m is the Brewster angle. Incoming rays with a angle larger than the Brewster angle are totally reflected and no refracted ray are emitted.

2.4 Simplification of the Rendering Equation

The rendering equation can be separated into different parts for simplicity. First the BRDF is split into two parts according to (12).

$$f_r = f_{r,s} + f_{r,d} \tag{12}$$

Where $f_{r,s}$ is the specular and $f_{r,d}$ is the diffusely reflecting part of the BRDF. The incoming radiance can also be split into different parts depending on the incoming lights direction, as seen in (13).

$$L_i = L_{i,l} + L_{i,c} + L_{i,d} \tag{13}$$

Where $L_{i,l}$ is the direct light contribution from the source, $L_{i,c}$ is the light contribution via specular reflections and $L_{i,d}$ is the light contribution from soft illumination.

By substituting f_r and L_i from the integration part of the rendering equation with (12) and (13), equation (14) is formed.

$$\begin{aligned} L_r = & \int_\Omega f_r L_{i,l} cos\theta_i + \\ & \int_\Omega f_{r,s}(L_{i,c} + L_{i,d}) cos\theta_i d\omega_i + \\ & \int_\Omega f_{r,d} L_{i,c} cos\theta_i d\omega_i + \\ & \int_\Omega f_{r,d} L_{i,d} cos\theta_i d\omega_i \end{aligned} \tag{14}$$

These parts can be calculated individually which makes the implementation simpler.

2.5 Rendering

This section describes how each of the terms from (14) are estimated. Since Monte Carlo is a recursive method some of the terms rely on each other. This had to be taken into account when implementing the rendering function. The terms could be computed completely separated. However, that would increase the rendering time since some computations would have to be evaluated more than once. Therefore, the parts are implemented to be evaluated simultaneously.

2.5.1 Direct Illumination

The first term from (14) indicates the direct illumination. This is simply calculated by sending multiple shadow rays toward the light source using the estimator seen in (15).

$$\langle L_{i,l} \rangle = \frac{AL_0}{M} \sum_{k=1}^{M} f_r \frac{cos\alpha_k cos\beta_k}{d_k^2} V_k \tag{15}$$

Where L_0 is the radiance from the light source, A is the light source area, M is the number of shadow rays, d is the distance to the light source and V_k is a visibility variable.

If the current point is visible, V_k is set to 1, otherwise it is set to 0. In the implementation multiple shadow rays were used to achieve a smoother transition between visible and hidden surfaces from the perspective of the light source.

2.5.2 Specular Reflection

The second term from (14) represents the light contribution of specular reflections. The specular light contributions estimator can be seen in (16).

$$\langle L_{i,c} \rangle = \frac{\pi L_0}{M} \sum_{k=1}^{M} f_r G_k \tag{16}$$

Where G_k is a geometric term. The effect of this part of the rendering equation can be seen on objects which are not diffuse reflectors or pure reflective materials. For example the effect is seen on transparent or glossy specular reflectors.

2.5.3 Caustics

The third part of (14) represent *caustics*. Caustic is a physical phenomena where light rays reflects or refracts by a curved surface or object. This results in bright spots on diffuse surfaces. Caustics are never computed with Monte Carlo sampling as it would be too expensive computationally. Instead it can be generated directly from a *caustic photon map* [5]. However, a photon map was not implemented during this project.

2.5.4 Indirect Diffuse Illumination

The last part of (14) indicates the light contribution from indirect diffuse reflections. The estimator for this is seen in (17).

$$\langle L_{i,d} \rangle = \frac{\pi}{M} \sum_{k=1}^{M} f_r L_0 \tag{17}$$

This is a major contributing part towards the experienced photorealism of a rendered image, as this part contributes towards diffuse surfaces which are hidden from the light source. In other words, surfaces that are in the shadow of another object. Without indirect diffuse light the shadows appear unrealistically dark compared to the rest of the scene.

Indirect diffuse light also introduce color bleeding which is the effect of transferring colored light from nearby surfaces according to their BRDFs.

2.6 Noise and Aliasing

When using the Monte Carlo scheme the renderer produce noise, this is because the Lambertian reflectors have a 75% chance to reflect the incoming ray. This means that there will be points next to each other that will have a drastic difference in their radiance compared to each other. This will be perceived as noise.

To counter this *multisampling* is introduced. Multisampling is a technique to reduce noise by dividing each pixel into sub pixels. The difference in radiance between points will remain. However, the points are now smaller and compressed in a dense area resulting in less apparent noise.

Another phenomena is *Aliasing*. Aliasing occurs when surfaces and their patterns are undersampled. *Supersampling* is introduced as a countermeasure for this. Supersampling is a extension of multisampling, where the rays are emitted at a random point of each subsample instead of its center. This results in a rendered image with reduced aliasing.

3 Results

In this section the results are shown in the form of images and tables. Features and their different impacts on rendering time will be highlighted. The First image illustrates all the features of the renderer, which includes Lamertian reflectors, Oren-Nayar reflectors, pure reflective surfaces and transparent objects. This image is seen in Figure 2. The figure also illustrates the noise reduction using the methods described in section 2.6.

Table 1 contains rendering times computed with an Intel Core i7-4790K CPU. Where the static rendering parameters were 400x400 pixels with 30 shadow rays per intersection.

Table 1: Rendering times for the images in Figure 2.

Samples per pixel	Rendering time
4	31s
16	2m:01s
64	7m:56s
256	30m:37s

To illustrate the difference between Lambertian and Oren-Nayar reflectors, four different images were rendered with different roughness values. These images can be seen in Figure 3. The first image has the roughness set to zero, which makes the material equal to a Lambertian reflector.

Table 2 show different rendering times for the images seen in 4. The renderings where computed with an Intel Core i7-4790K CPU. Where the static rendering parameters were 400x400 pixels with 30 shadow rays per intersection and 16 samples per pixel.

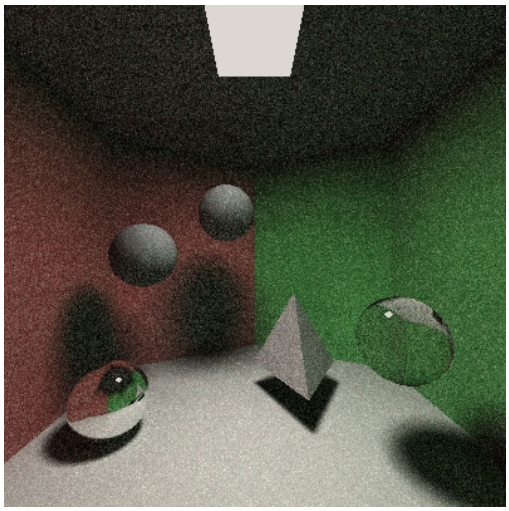
Table 2: Rendering times for the images in Figure 4.

Material	Rendering time
Pure reflective	1m:02s
Lambertian reflector	1m:04s
Transparent	1m:48s

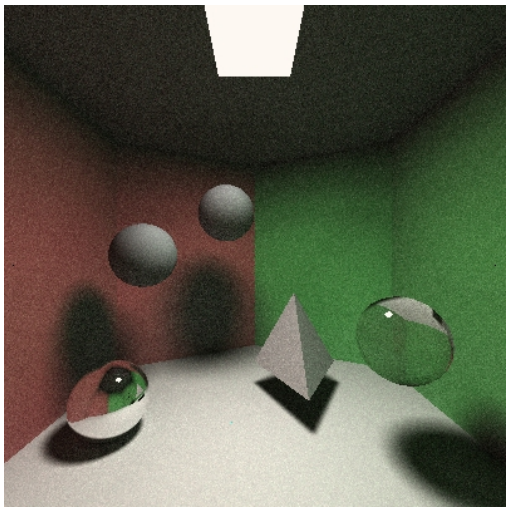
In Figure 5 four images can be seen, where the scene was rendered with a different number of shadow rays per intersection. The corresponding rendering times is seen in Table 3 with the static parameters set to 400x400 pixels with 64 samples per pixel using a Intel Core i7-4790K CPU.

Table 3: Rendering times for the images in Figure 5 where the number of shadow rays vary.

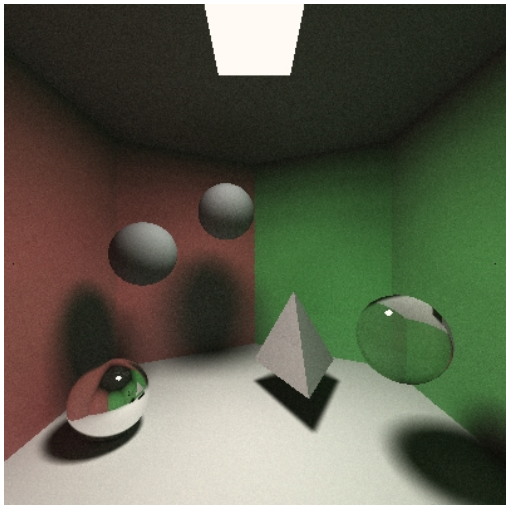
Number of shadow rays	**Rendering time**
1	43s
2	1m:03s
4	1m:31s
8	2m:28s



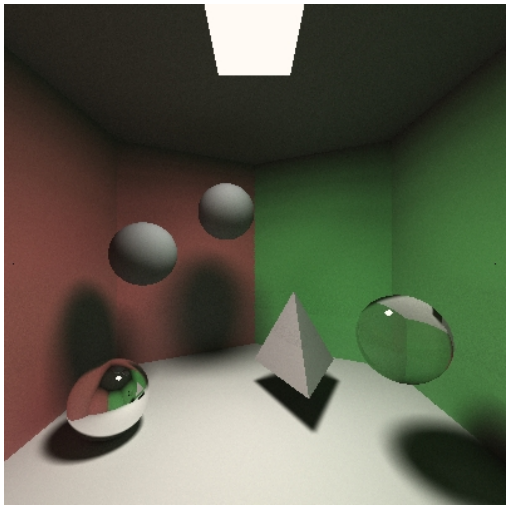
(a) 4 samples per pixel



(b) 16 samples per pixel

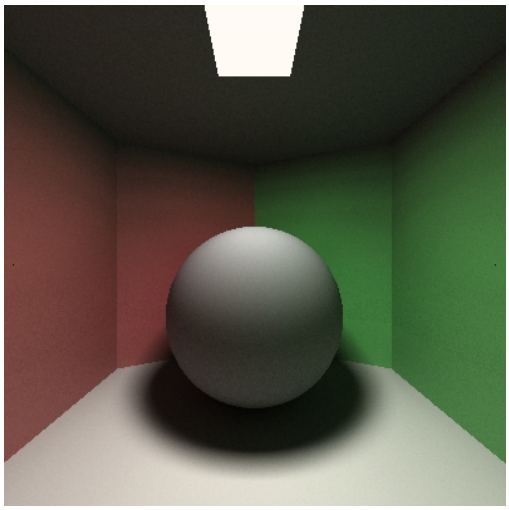


(c) 64 samples per pixel

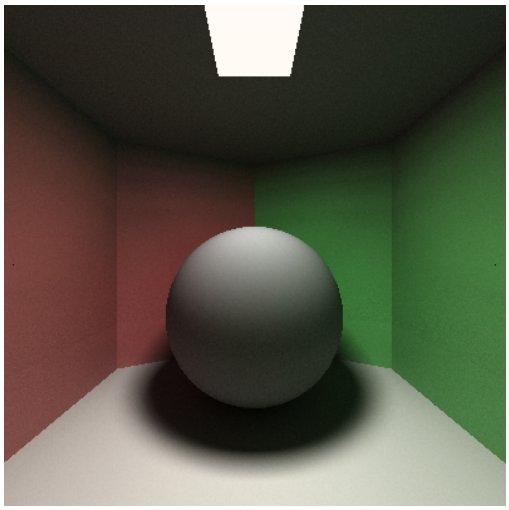


(d) 256 samples per pixel

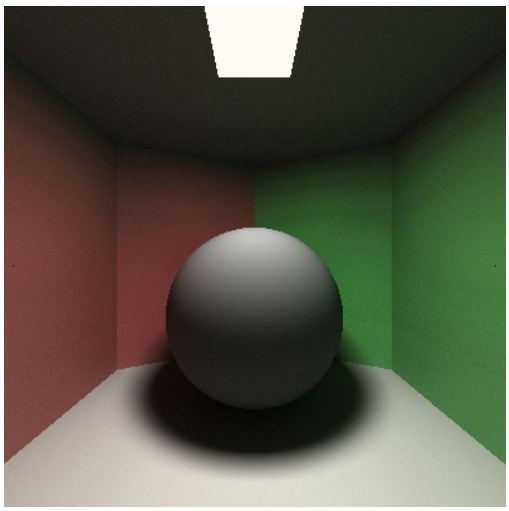
Figure 2: Rendering of a scene with varying number of samples per pixel. The images are rendered with 400x400 pixels with 30 shadow rays per intersection



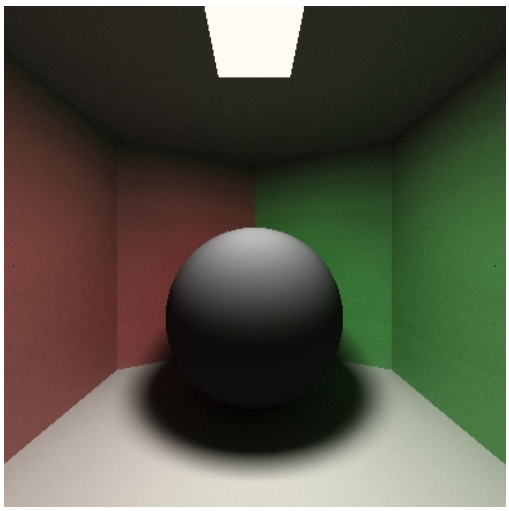
(a) Oren-Nayar BRDF with roughness set to 0



(b) Oren-Nayar BRDF with roughness set to 0.6

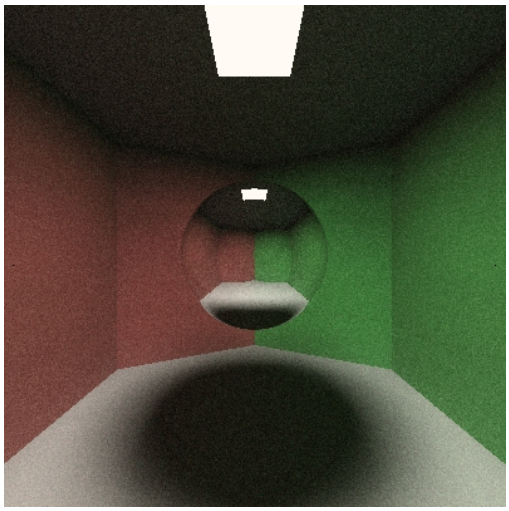


(c) Oren-Nayar BRDF with roughness set to 0.8

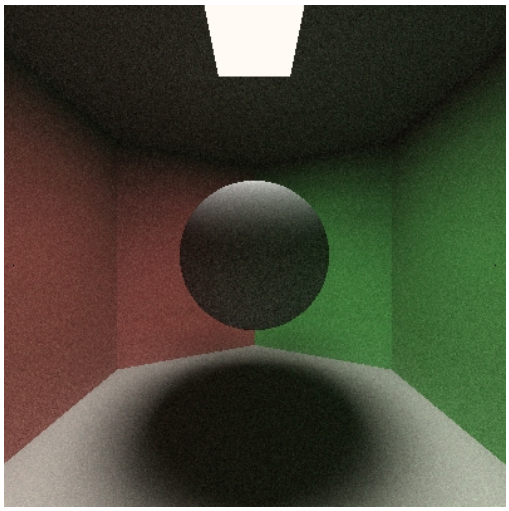


(d) Oren-Nayar BRDF with roughness set to 1

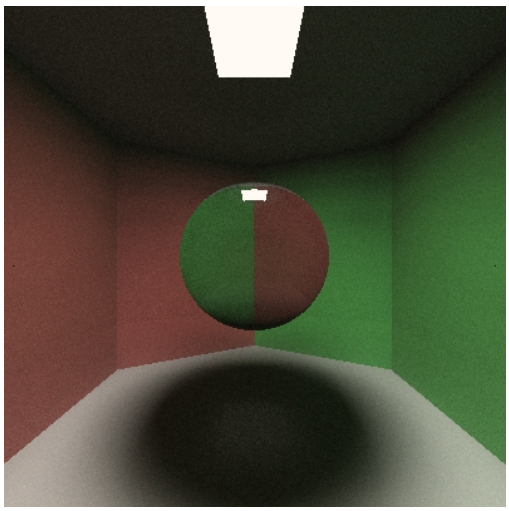
Figure 3: Rendering of a sphere with a Oren-Nayar BRDF with varying roughness values



(a) Pure reflective sphere

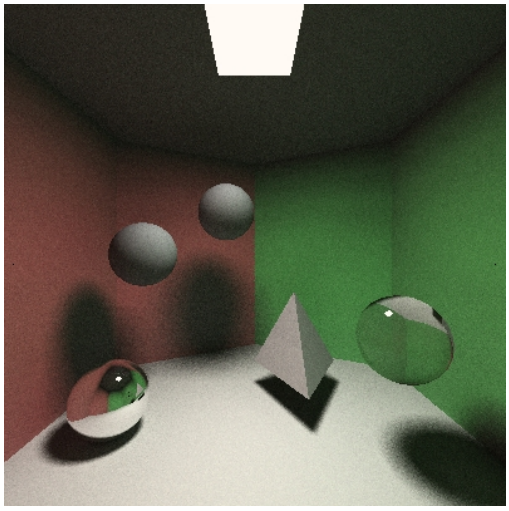


(b) Lambertian sphere

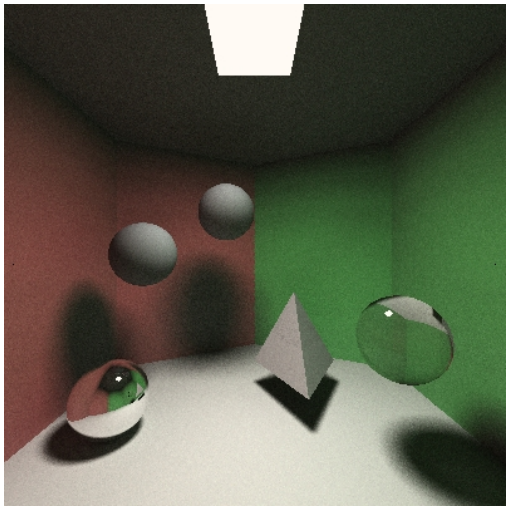


(c) Transparent sphere

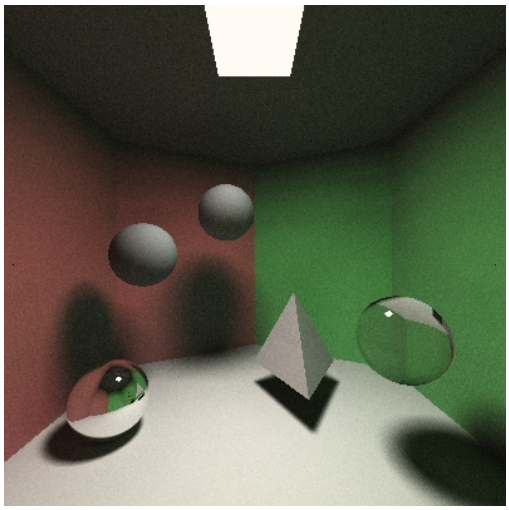
Figure 4: Rendering of a sphere with different material to compare the rendering times. The result can be seen in Table 2



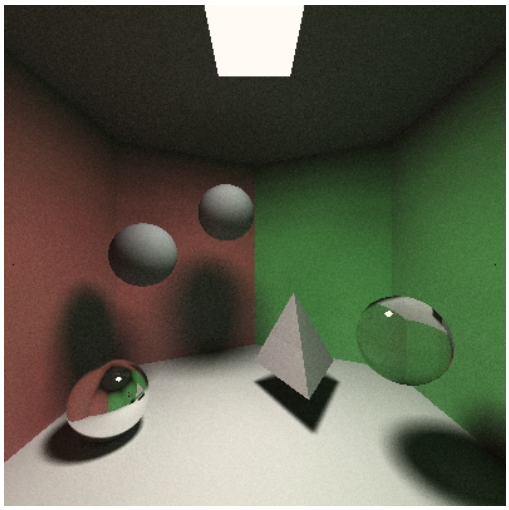
(a) Scene with 1 shadow ray per intersection



(b) Scene with 2 shadow ray per intersection



(c) Scene with 4 shadow ray per intersection



(d) Scene with 8 shadow ray per intersection

Figure 5: Rendering of a 400x400 scene with 64 samples per pixel

4 Discussion

Implementing a Monte Carlo Ray Tracer which is based on physical light is not an easy task. Debugging combined with long computational time does not go well together. Therefore, the key to this project was the solutions for optimizations and approximations. Once multiprocessing was introduced through OpenMP, the debugging time decreased significantly, especially when used on a four core CPU.

The result from the shadow ray test evaluated in Figure 5 and Table 3 implies that eight shadow rays are enough to produce smooth shadows. This is confirmed by comparing the minimal difference of Figure 2c and Figure 5d, where the difference between the number of shadow rays are 22. Increasing the number of shadow rays after eight only seems to increase the computational time whilst the image quality is marginally improved. This concludes that a unnecessary amount of shadow rays were used for the other images, which was 30 shadow rays per intersection.

An optimization problem occurred when introducing the transparent object. For transparent objects some rays do not refract rays, this is because the angle is larger than the Brewster angle, which is the expected behavior. However, this can lead to an infinite amount of reflecting rays which causes stack overflow. To counter this a stopping condition for the transparent reflections was set to a large number. If the value is large enough, it is unlikely for the diffuse reflectors to ever reach this value using Russian Roulette. This Keeps the ray tracer unbiased. Unfortunately, this led to another problem.

When intersections with transparent objects are evaluated, some trees would now have a significant difference between their depths. The multiprocessing method implemented is not optimized for this case leading to a bottleneck in the CPU. This is one reason why the computational time of the transparent objects is longer compared to the other objects in Table 2. This can also be confirmed by evaluating the CPU usage percentage during the rendering of a transparent object, compared to a diffuse reflector. For a transparent object the CPU will fluctuate between low percentages and a high percentage, where it in the best case should be stable at a high percentage similar to the diffuse objects.

Further development would be to introduce caustics, which can be achieved by implementing photon mapping as mentioned in section 2.5.3. Caustic would greatly contribute towards the perceived photorealism of the ray tracer. It would also be possible to introduce GPU programming for further optimization.

The reliability of the achieved photorealism can be improved. This is the disadvantage of using a lesser known benchmarking scene. By using a scene such as the traditional Cornell Box, there would be several global illumination renderings to compare the results of this paper to. Another solution would be to recreate our scene in real life to realise its accuracy.

The knowledge gained by working on this project will hopefully function as a solid foundation for future studies in computer graphics. It is interesting to see that photorealism can be achieved by using scientific papers from the 80:s. It is also interesting to have experienced the amount of computations and the required time to achieve global illumination. Nowadays, even real-time ray tracing exists, which goes to show how far processing power paired with optimization has come.

References

[1] J. Turner Whitted, *An Improved Illumination Model for Shaded Display*, 1980, Retrieved: October 24, 2019, https://www.cs.drexel.edu/~david/Classes/Papers/Whitted80.pdf

[2] Matt Pharr, Wenzel Jakob and Greg Humphreys, *A Brief History of Physically Based Rendering*, 2019, Retrieved: October 28, 2019, http://www.pbr-book.org/3ed-2018/Introduction/A_Brief_History_of_Physically_Based_Rendering.html

[3] James T. Kajiya, *The Rendering Equation*, 1986, Retrieved: October 28, 2019, http://www.cse.chalmers.se/edu/year/2011/course/TDA361/2007/rend_eq.pdf

[4] Donald P. Greenberg, John R. Wallace and Michael F. Cohen, *A TwoPass Solution to the Rendering Equation: A Synthesis of Ray Tracing and Radiosity Methods* 1987, Retrieved: October 31, 2019, https://pdfs.semanticscholar.org/ce6f/0a8b678a9d23a40df43070620ea7daa845a8.pdf

[5] Henrik Wann Jensen, *Global Illumination using Photon Maps*, 1996, Retrieved: October 24, 2019, http://graphics.ucsd.edu/~henrik/papers/photon_map/global_illumination_using_photon_maps_egwr96.pdf

[6] Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg and Bennett Battaile, *Modelling the Interaction of Light Between Diffuse Surfaces* 1984, Retrieved: October 31, 2019, https://www.cs.rpi.edu/~cutler/classes/advancedgraphics/S12/papers/goral_radiosity_84.pdf