Monte Carlo Ray Tracer

Johan Forslund
Pontus Arnesson

November 2019

1 Abstract

When rendering images we often strive for photo realism. In order to solve this, we need to solve the global illumination problem, which means simulating how light behaves in the real world. In this report we introduce the concepts of *Monte Carlo ray tracing* and show the results of an implementation, including benchmarks and a discussion about the ways to optimize the results.

We begin by introducing the rendering equation and how ray tracing solves this in an elegant way. We continue with describing how *radiance* is transported through the scene and ends up at different surfaces, giving the resulting image. Theory that is discussed in the report includes *supersampling*, *shadow rays*, an approximation of *Fresnel's equation*, *Snell's law* and more.

The results are presented using both implicit and polygonal objects, with materials including perfect mirrors, transparent objects, *Lambertian* reflectors and *Oren-Nayar* reflectors.

2 Introduction

Computer rendered images are a requirement in many fields. People working with visualization for example often need computer graphics to visualize the information they want to show. Likewise, it is used for games, movies and lots of other medias. The most classic way of rendering images is with a local lighting model. With this model, the light for each geometry in the scene is calculated independently. Thereby, light reflections between geometries will not be taken in account, which results in loss of useful light information. This is however an efficient method and is for example often used in real time rendering, but it does not really produce photo realistic images.

To render photo realistic images with a computer, we need to simulate real world lighting within the computer program and thereby add the reflected light between objects. In other words: we need global illumination. By solving the global illumination problem we can have much more sophisticated scenes with for example color bleeding, caustics and mirrors which often is a necessity to create a photo realistic image. The global illumination problem can be represented mathematically using the rendering equation (1) which (conceptually) is a way to describe light information at a point, given information about all the light that will reach that point from other points. It is thereby a recursive equation, which different rendering techniques tries to solve.

$$L(x \to w_{out}) = L_e(x \to w_{out}) + \int f_r(x, w_{in}, w_{out}) L(x \leftarrow w_{in}) cos\theta_{in} d\omega_{in} \quad (1)$$

One solution to the global illumination problem is using ray tracing. This means following rays from a light source to the spectating eye as the ray reflects of different materials. The goal is to have an algorithm where an increase in computing power makes the image converge to a photo realistic image. In the ideal case, we could use an infinite amount of computing power, which would produce images that look just like real world photos. However, due to restrictions in computers we will have to stick with images that are close to photo realism.

In this report we present a particular ray tracing technique called Monte Carlo ray tracing, though this is just one of several different methods to solve the same problem.

2.1 Whitted ray tracing

Whitted ray tracing is one way to simulate how the light rays travel through a scene. This is a bias method as the light ray always gets terminated directly on diffuse surfaces. The idea is to launch a ray from the spectating eye into the scene instead of sending it from the light source. The ray then either intersects with a diffuse object and gets terminated, or hits a transparent object or perfect mirror. The reflected and refracted rays from mirrors and transparent objects get traced recursively further in a given direction depending on the material of the object. This recursion creates a tree of rays. When the ray finally intersects with a diffuse object, the recursion stops and the light intensity from the leaf node traverses up the the tree and gets added to the parent nodes. In Whitted ray tracing, the approach is to use a point light source. Since the ray is launched from the eye instead of the light source, the possibility for the ray to hit the point light source therefore is minimal. Because of this, a ray is sent from the point directly to the light source at every intersection to see if that point should be in shadow or not. This causes very hard shadows since the rays cannot be reflected diffusely, and the pixels are either totally visible or completely blocked from the point light. This can be seen in Figure 1. It can also be seen that there is no color bleeding between objects. This is due to the fact that no diffuse reflections occurs in the scene at all.

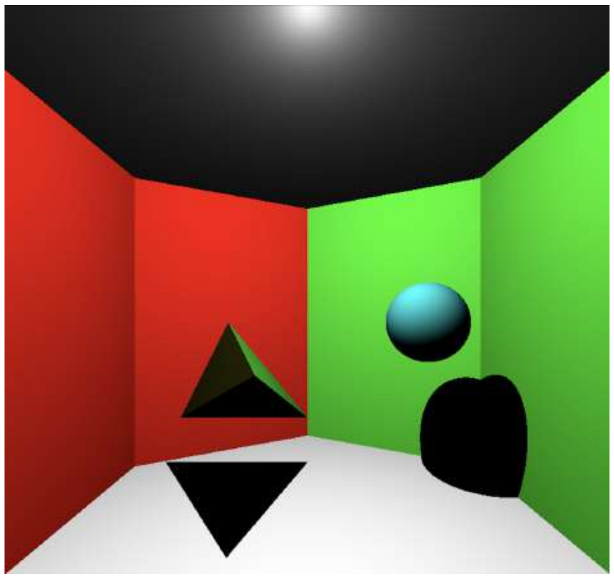

Figure 1: Scene rendered using Whitted ray tracing

2.2 Monte Carlo integration

For objects in the scene that are not perfect reflectors, we will in theory have lots of outgoing rays for each intersection point. For example, a type of diffuse surface called Lambertian will spread the outgoing light into all directions. This would be an impossible task to compute for a computer if we want to trace every outgoing ray. Instead, the Monte Carlo method that is known from statistics can be used to simulate a non-perfect reflector without adding bias to the output.

The idea is to draw random outgoing ray directions for each intersection. The ray is then recursively reflected through the scene until it hits a light source or another stopping condition is activated. To avoid the case where a ray never reaches a light source we use Russian Roulette to terminate some of the rays in a random fashion but without introducing bias.

As opposed to Whitted ray tracing, the Monte Carlo integration can have soft shadows as area light sources can be used. Instead of having just one ray going directly to the light source to check for visibility, multiple rays are often used. This means that the point can be partly in shadow, since only some of the rays are blocked, and hence give the illusion of soft shadows.

Figure 2 shows the light effects that are being produced by Monte Carlo ray tracing. The randomness of this technique is the reason we have lots of noise in the image. We see how the addition of diffuse materials (on the walls) results in colored light on the roof. The left side of the roof have a red tint to it while the right side has a more green tint. This is the result of rays getting reflected from the wall onto the roof. We can also see how the area light source produces soft shadows to the right of the cube and below the sphere. The shadows are also noisy, which comes from the fact that only few rays have been sent to the light source from the shadow points.

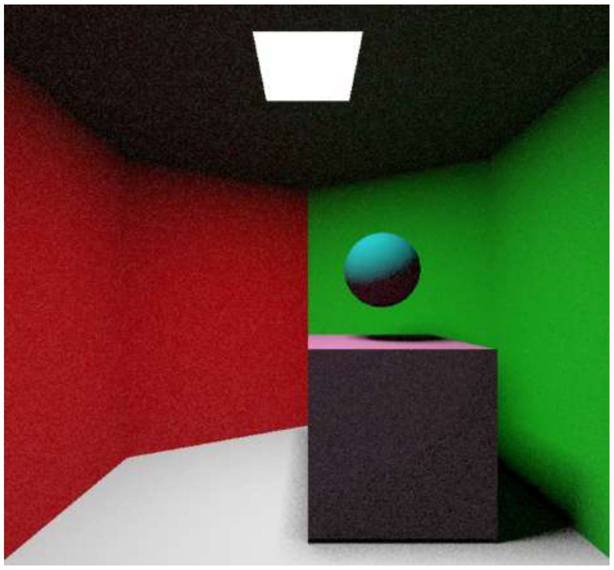

Figure 2: Scene rendered using Monte Carlo ray tracing

For this report, we have implemented our own Monte Carlo ray tracer and the results from the implementation is presented and discussed at the end of the report.

3 Background

The basic aim of the ray tracing technique is to acquire an intensity for each pixel in the image plane that will make the resulting image as close to reality as possible. For this, several physical phenomenons has to be taken into account and, in some cases, be approximated to make this possible for a machine to compute. In this section we will further describe how this can be done.

3.1 Hemispherical coordinates

For every point the rays hit in the scene, one could imagine a hemisphere around that point. As can be seen in Figure 3, there are two different angles where φ is the *azimuth* angle and θ is the *inclination* angle. Here $\varphi \in [0, 2\pi[$ and $\theta \in [0, \frac{\pi}{2}]$ This makes directions easier to deal with since it reduces the dimension to two.

One property that can be calculated with the help of the local hemisphere is the *solid angle*. The solid angle Ω is the surface element A projected by an object on the hemisphere with radius r. The solid angle has the unit *Steradian* and $\Omega = \frac{S}{r^2}$, $\Omega \in [0, 2\pi]$.

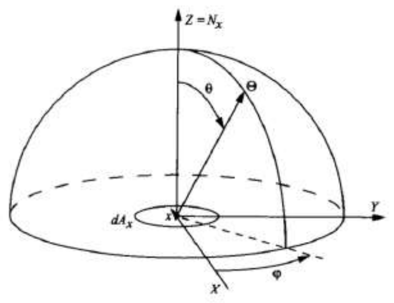

Figure 3: Illustration of the hemisphere around a point x

3.2 Radiometric units

To be able to compute the global illumination of a scene there are different radiometric units to take into account. The first one is *flux* which describes how much energy is emitted or absorbed by a surface A in a given time interval [3]. Flux has the unit Watts (Joules / seconds). Other radiometric units are:

- *Radiance* L is a measure of flux per projected area per solid angle on a hemisphere around the point x.
- *Irradiance* E is the incoming differential flux onto a point x. $E = \frac{d\theta}{dA}$.
- *Radiosity* M is the outgoing differential flux from a point x. $M = \frac{d\theta}{dA}$.

3.3 Rays

Each ray that gets sent out from the eye carry *importance*. Importance describe how important that ray is for the resulting image. The importance for the ray with a start point in the eye is always one, and when the depth of the ray increases (how many times the ray has bounced) the importance decreases more and more. This importance is then used to calculate how much of the radiance the ray will receive when it intersect with a surface. When doing calculations in the ray tracer, we always assume that the radiance of the ray is the same at the start and at the en point of the ray.

3.3.1 Supersampling

Supersampling is when multiple rays are sent through the same pixel. Either the rays can be distributed evenly in the pixels by so called sub pixels, or randomly by ray randomization. Supersampling is used to give the image a more smooth look and reduce aliasing that can appear in the render. The use of evenly distributed supersampling is less effective on aliasing but also less noisy than the use of ray randomization.

3.4 Rendering equation and BRDF

The rendering equation (1) was quickly introduced in Section 2 but now we explain the equation in terms of radiance, where the equation is defined by:

- $L(x \rightarrow w_{out})$ describes radiance emitted from a point x into the direction w_{out}
- $L_e(x \rightarrow w_{out})$ is radiance coming from light source contribution at a point x, going into the direction w_{out}
- $f_r(x, w_{in}, w_{out})$ is the bidirectional reflectance distribution function (BRDF), explained in the next segment
- $L(x \leftarrow w_{in})$ is radiance coming from a direction w_{in} into the point x
- $cos\theta_{in}$ is a geometrical factor that scales the integrand by the fact that a large inclination angle θ_{in} will give less radiance at the point.

We integrate $f_r(x, w_{in}, w_{out})L(x \leftarrow w_{in})cos\theta_{in}$ over all possible angles ω_{in} on the hemisphere to capture all the radiance that will flow into the point x. It is the BRDF $f_r(x, w_{in}, w_{out})$ that determine how much of the potential radiance will reach the point x. This distribution function is defined by the material properties of the object that point x is located on. The BRDF for some materials (such as perfect mirrors) can easily be described mathematically, while others may be much more complex. There are ways to measure the BRDF for real life objects, for example by using luminance and illuminance meters [2], which is useful when we want to render scenes with arbitrary materials.

3.5 Material properties

As described in the previous section, the BRDF is defined by the material properties of the object. In this report we will describe perfect mirrors, transparent materials, Lambertian reflectors and Oren-Nayar reflectors.

3.5.1 Perfect mirror

A perfect mirror is one of the easiest material to describe mathematically. Figure 4 shows the concept quite clearly. An incoming ray will get reflected according to the perfect reflection law, which means the incoming and outgoing angle, relative to the normal, will be the same.

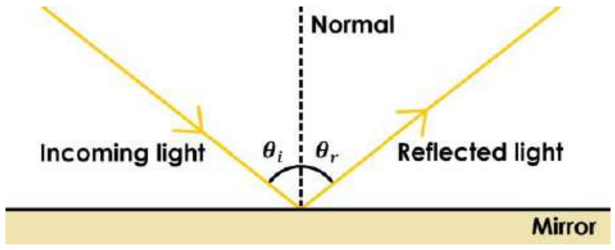

Figure 4: A ray reflected perfectly.

Mathematically, this BRDF can be described with a Dirac delta function as the mirror will only emit radiance into one direction.

$R(\omega_{in})$ is the direction along which ω_{in} is reflected at point x.

3.5.2 Transparent materials

Transparent materials are similar to perfect reflection materials in the sense that there is always one ray going in the perfect reflected direction. The difference is that there is also one refracted ray that is going in a direction into the object. This direction is calculated by *Snell's law* (2) and can be seen in Figure 5.

$$\frac{\sin\theta_i}{\sin\theta_{refr}} = \frac{n_{refr}}{n_i} \tag{2}$$

There is, however, a case called total internal reflection where the incoming ray gets completely reflected. This happens when the ray goes from a more dense to a less dense medium and the incident angle is more than the critical angle. The critical angle is computed by $\sin^{-1}\frac{n_i}{n_{refr}}$.

The distribution of radiance between these refracted and reflected rays can be calculated by the *Fresnel's equation*. Where the total reflection R consists of the electric field vector that points out of the plane R_s (4), and the electric field vector in the plane R_p (5).

$$R = \frac{R_s + R_p}{2} \tag{3}$$

$$R_p = [\frac{n_i cos\theta - n_{refr}\sqrt{1-([n_i/n_{refr}]*sin\theta_i)^2}}{n_i cos\theta + n_{refr}\sqrt{1-([n_i/n_{refr}]*sin\theta_i)^2}}]^2 \tag{4}$$

$$R_s = [\frac{n_i\sqrt{1-([n_i/n_{refr}]*sin\theta_i)^2} - n_{refr}cos\theta_i}{n_i\sqrt{1-([n_i/n_{refr}]*sin\theta_i)^2} + n_{refr}cos\theta_i}]^2 \tag{5}$$

However, this method is computationally expensive. Instead the calculations are approximated by *Schlick's law* (6) where θ is the angle between the incoming ray and the surface normal, and R_0 is calculated by equation (7).

$$R(\theta) = R_0 + (1 - R_0)(1 - cos\theta)^5 \tag{6}$$

$$R_0 = (\frac{n_i - n_{refr}}{n_i + n_{refr}})^2 \tag{7}$$

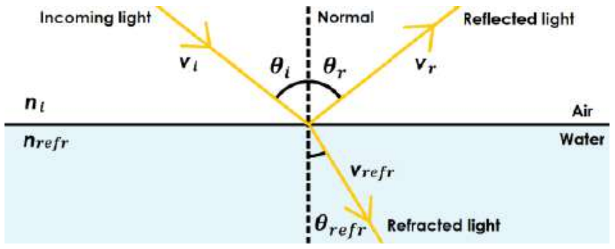

Figure 5: A ray being reflected and refracted by Snell's law.

3.5.3 Lambertian reflector

For a Lambertian reflector, the radiance gets emitted evenly into all possible directions. Thereby, the BRDF becomes constant, see (8).

$$f_r(x, \omega_{in}, \omega_{out}) = \frac{\rho}{\pi} \tag{8}$$

ρ is called the reflectivity of the surface [5]. The division by π is to factor for energy conservation.

Though this is the correct description in theory, it would require an infinite amount of computing power to follow the rays for every possible outgoing direction. This is where statistics come into the picture, more specifically Monte Carlo integration.

Instead of following all outgoing rays for a Lambertian surface, we use probability theory to draw random outgoing rays from a well chosen probability distribution. We can define a probability distribution function (PDF) $p(x)$ that mathematically describes a probability distribution for a random variable. The PDF has to fulfill the conditions (9) and (10).

$$p(x) \geq 0 \tag{9}$$

$$\int_{-\infty}^{\infty} p(x)dx = 1 \tag{10}$$

When we have defined the probability distribution function we can draw random numbers with that specific PDF using the relation between the PDF and its associated cumulative distribution function (CDF) [4]. A CDF $P(x)$ is the probability that a random variable takes a value less than or equal to x, as (11).

$$P(x) = \int_{-\infty}^{x} p(\tilde{x})d\tilde{x} \tag{11}$$

With the CDF, we can draw random numbers that follows a PDF, see (12).

$$x_i = CDF^{-1}(y_i) \tag{12}$$

y_i is a random number drawn from a uniform PDF and x_i is the random number that follows the desired PDF. CDF^{-1} is just the inverse of the CDF, which always will be defined since the CDF is strictly increasing.

3.5.4 Oren-Nayar reflector

The Lambertian model is adequate in most cases. It is though quite deficient in accurately describing the reflected radiance distribution for both smooth and rough surfaces [7]. The Oren-Nayar model improves this by predicting the reflectance for rough surfaces.

The local geometry is assumed to be microfacets arranged in v-grooves. These are distributed in various orientations with the microfacets approximated to be lambertian reflectors. The roughness of the surface is then specified of a probability function depending on the distribution of the facets slopes, in this case a Gaussian distribution. The standard deviation σ of the Gaussian distribution is therefore a measure of the roughness of the surface. The resulting BRDF can hereby be seen in (13) with $\varphi_{in}, \omega_{in}$ and $\varphi_{out}, \omega_{out}$ are the directions of the incoming- and exitant rays. A and B are calculated by (14) while α and β is defined by $\alpha = max(\theta_{in}, \theta_{out})$, $\beta = min(\theta_{in}, \theta_{out})$.

$$f_r(x, \omega_{in}, \omega_{out}) = \frac{\rho}{\pi}(A + Bmax(0, cos[\varphi_{in} - \varphi_{out}])sin(\alpha)sin(\beta)) \tag{13}$$

$$A = \frac{\sigma^2}{2(\sigma^2 + 0.33)}, B = \frac{0.45\sigma^2}{\sigma^2 + 0.09} \tag{14}$$

3.6 Geometries and their ray intersections

We will describe two types of objects in this report: implicit and polygonal. Both of these types of objects can represent most scenes quite well since we can use polygonals (e.g triangles) to create complex objects and complement the scene with implicit objects (e.g spheres) when possible. For the ray tracing routine to work we need a way to detect whenever a ray hits one of these geometries, and also information about the reflected ray direction.

3.6.1 Triangle

Most of the objects in a general scene will most likely be composed by triangles and we thereby need an efficient method to determine when a ray intersects those triangles. One such method is the Möller-Trumbore intersection algorithm, which has the advantage that the plane equation does not have to be calculated for each intersection test [8]. The algorithm is widely used and is possible the fastest algorithm for calculating ray intersections with triangles.

A triangle is defined by three points: $\boldsymbol{v_0}$, $\boldsymbol{v_1}$ and $\boldsymbol{v_2}$. We use the barycentric coordinates u and v to define the triangle as (15), where $u \geq 0$ and $v \geq 0$.

$$u + v \leq 1 \tag{15}$$

We can then define a point on the triangle as (16)

$$T(u, v) = (1 - u - v)\boldsymbol{v_0} + u\boldsymbol{v_1} + v\boldsymbol{v_2} \tag{16}$$

Any ray in the scene is defined as (17)

$$x(t) = \boldsymbol{p_s} + t(\boldsymbol{p_e} - \boldsymbol{p_s}) \tag{17}$$

If we equal (16) and (17) we get the intersection point as (18).

$$(1 - u - v)\boldsymbol{v_0} + u\boldsymbol{v_1} + v\boldsymbol{v_2} = \boldsymbol{p_s} + t(\boldsymbol{p_e} - \boldsymbol{p_s}) \tag{18}$$

We can then solve for t and insert it into (17) to find the intersection point.

3.6.2 Sphere

Ray intersections with a sphere is calculated in a similar way to triangles, but here we define a surface point $\boldsymbol{x}$ with a radius r and a center point $\boldsymbol{c}$, see (19).

$$||\boldsymbol{x} - \boldsymbol{c}||^2 = r^2 \tag{19}$$

In this case we define a ray in the scene as (20).

$$\boldsymbol{x} = \boldsymbol{o} + d\boldsymbol{l} \tag{20}$$

$\boldsymbol{o}$ is the starting point and $\boldsymbol{l}$ is the normalized direction. If we insert (20) into (19) and solve for d we can then get the intersection point by inserting d into (20).

3.7 Shadow rays

As discussed in 2.1, when a ray hit a point in the scene a new ray is sent to the light source. This ray is called a shadow ray and is used, as described before, to see if that point should be in shadow or not. This is done by the ray intersection methods discussed in the previous section using the shadow ray.

Another use for the shadow ray is to determine how bright that point should be depending on the orientation relative to the light source. In Whitted ray tracing this is done by calculating the angle θ between the shadow ray and the normal of the point. If the angle is 0 i.e the point facing directly towards the light source, then we take full contribution from the light. If, on the other hand, the angle θ is more than $\frac{\pi}{2}$ or less than $-\frac{\pi}{2}$ the light contribution is zero at that point.

In Monte Carlo ray tracing the approach is a bit different. Since we are sending multiple shadow rays at random positions on the light source, the calculation for direct light contribution has to be done differently. The equation (21) shows how the direct light is computed at the point x_N in the direction $-\omega_{N-1}$ with Monte Carlo integration. A represents the area and L_0 the radiance of the light source. M is the number of shadow rays used, d_k is the length of the shadow rays and V_k is 0 if the point is in shadow and 1 if it is visible. α_k and β_k is the inclination angles of ω_{in} relative to the normals at the point and the light source.

$$L_D(x_N \to -w_{N-1}) = \frac{AL_0}{M} \sum_{k=1}^{M} f_r(x_N, \omega_{in,k}, -\omega_{N-1}) \frac{cos\alpha_k cos\beta_k}{d_k^2} V_k \tag{21}$$

4 Results

To visually demonstrate the theory described in this report, we show the results of our own ray tracer in this section. We compare different parameters to see what effect they give to the final output and how the performance is affected.

4.1 The scene

The scene in our implementation consists of different geometries and materials, with walls that encapsulate the room. The purpose of the encapsulation is that no ray will ever go outside of the scene, and therefore always get a light intensity at the end of every ray path. The basic structure of our scene can be seen in Figure 6 where the spectating eye (1) is where all the rays originate from. The rays are then (in the simplest case) sent to the middle of each pixel of the image plane and from there continuing into the scene.

The materials and objects used in the scene (as can be seen in Figure 6) are the following:

- Diffuse Lambertian material on walls (except one), roof, box (2) and tetrahedron (3)
- Mirror material (perfect reflection) on the small sphere (5) and one wall
- Transparent material on the big sphere (4)

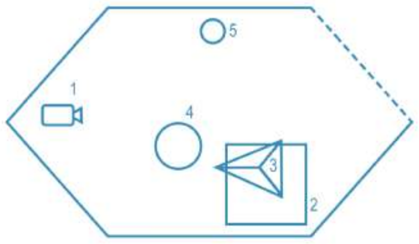

Figure 6: Basic structure of the scene used in the ray tracer

4.2 Final image

Different parameters can always be tweaked to improve the quality of the image. Since this is a routine aiming for photo realism, there really is not any limit where the parameters give best result. But for this image we have used parameters which gave a reasonable render time. However, at this point there is not a significant increase in quality if we would have used even more demanding parameters, since the image is quite small and the human eye would not spot further differences that much. The image is shown in Figure 7.

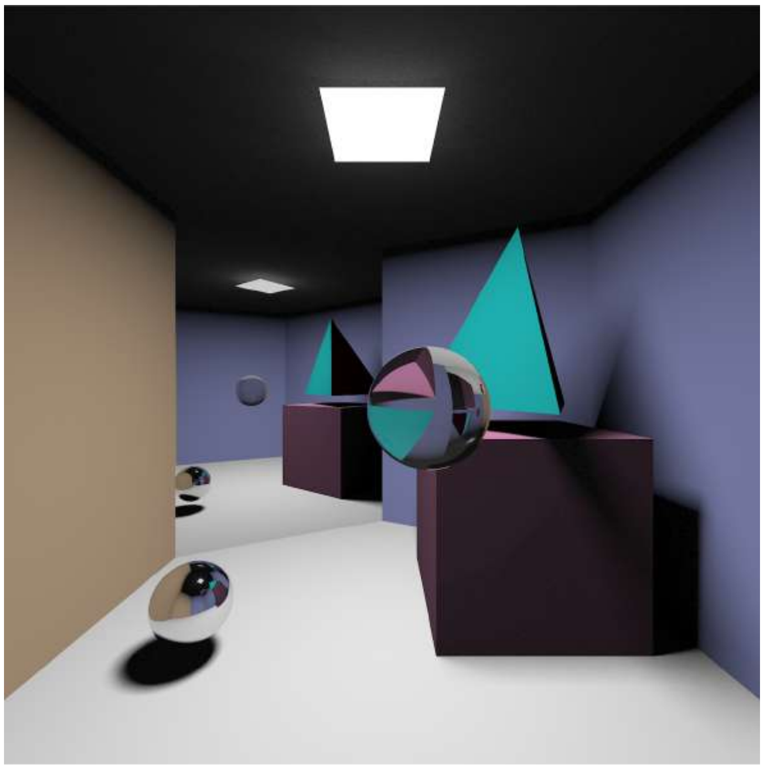

Figure 7: Final image

We see how the ray tracing method gives us reflections between different objects, most obvious case is the mirror sphere.

4.3 Supersampling

There are some obvious differences in the resulting image when varying the *samples per pixel*(SPP). Figure 8 shows the difference between a render using one SPP (left hand side) and using four SPP (right hand side). In the image with one SPP one can clearly see that the box and walls are more noisy, and the overall smoothness of the image is much worse.

The supersampling in this image is done with sub pixels, and therefore reduce the noise more than if randomization would be used. Since there were no real aliasing in the image we decided to keep the evenly distributed supersampling to get the best result regarding noise. The SPP could be increased more, but the quality of the image doesn't increase in proportion with the time it takes to do the rendering.

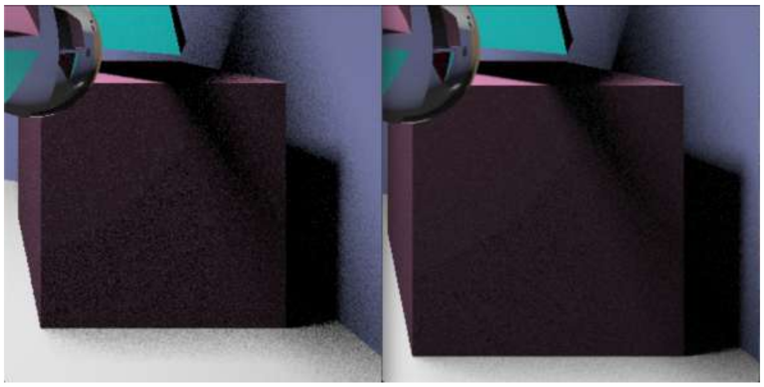

Figure 8: Difference between 1 SPP (left) and 4 SPP (right)

4.4 Depth

The depth of the ray implies how many times the ray has bounced. In Figure 9 we can see that the image to the right has more depth when looking at the reflection of the transparent sphere itself in the mirror (marked by the red square). In the right image we can see multiple colors of diffuse surfaces while in the left only one color is shown. This is because the ray terminates much earlier in the case when we only use three bounces.

However, this is mostly important when having a lot of mirror and transparent materials. The total difference of the image is not changed significantly apart from the specific case described above.

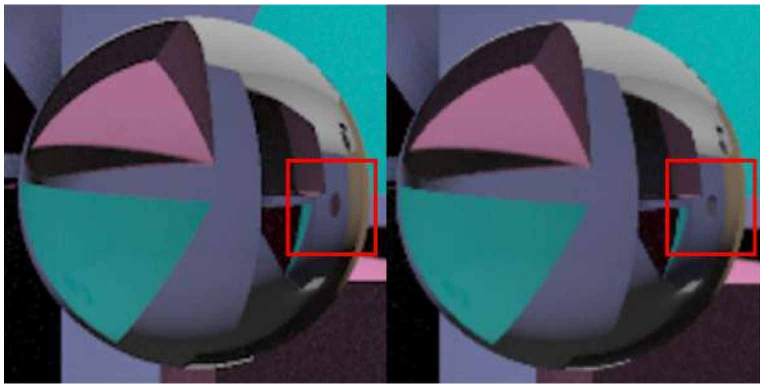

Figure 9: Difference between ray depth 3 (left) and ray depth 8 (right)

4.5 Lambertian and Oren-Nayar

In Figure 10 we see two spheres where the left sphere has a Oren-Nayar reflecting material and the right sphere has a Lambertian reflecting material. The difference between the reflectors are not large but we can see on the right image that the sphere appears to have a more smooth surface (disregarding noise) with its edges more darker than the brighter center area. The sphere in the left image has a more constant lighting over the whole sphere.

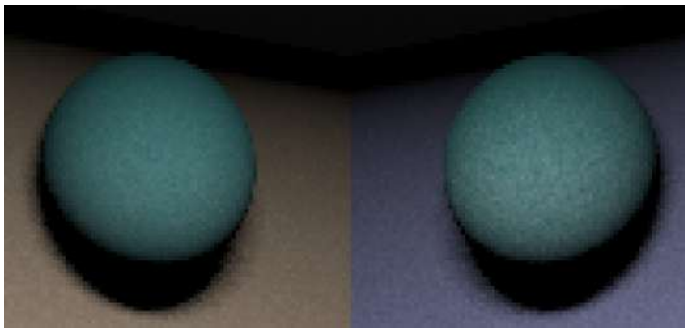

Figure 10: Difference between Oren-Nayar reflector (left) and Lambertian reflector (right)

4.6 Shadow rays

The number of shadow rays sent at each intersection will contribute to how smooth the shadows are in the scene. In Figure 11, the left image was rendered with 1 shadow ray per intersection while the right image was rendered with 30 shadow rays per intersection. The box in the left image has a harsh shadow with lots of noise while the box in the right image has a much smoother shadow.

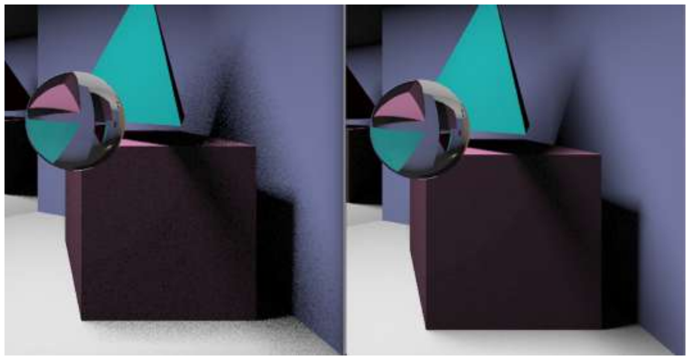

Figure 11: Difference between 1 shadow ray (left) and 30 shadow rays (right)

4.7 Russian Roulette constant

When applying the Russian Roulette routine, we use a probability constant α to vary the probability that a ray gets terminated at diffuse surfaces. We see in Figure 12 that a too low number introduces noise into the picture as we terminate too many rays at a low depth.

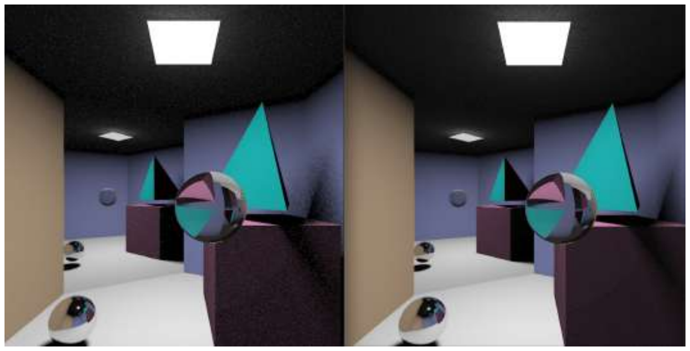

Figure 12: Difference between $\alpha = 0.01$ (left) and $\alpha = 0.5$ (right)

4.8 Benchmarks

Previously we have seen how different parameters for the ray tracing routine affects the visual result. Here we look at the time needed to render the image with different parameters. The results are shown in Table 1.

SPP is samples per pixel. Shadow rays is how many shadow rays are sent to the light source at each intersection point. RR constant is the constant that is used during the Russian Roulette routine to determine the probability that a ray gets terminated.

We can draw some direct conclusions from the results. Since the shadow rays won't get reflected further (no new rays are needed), it will only slow down the process linearly. An increase in 30x shadow rays increased the time by 19x. The same conclusion can be drawn from the SPP increment.

Table 1: Benchmark results

SPP	**Shadow rays**	**RR constant**	**Time (min)**
16	1	0.25	7
16	30	0.25	132
1	20	0.25	5
4	20	0.25	23
4	20	0.01	5
4	20	0.9	51

5 Discussion

We have seen how a ray tracing routine can be used to solve the global illumination problem. It is a time consuming routine but has the advantage that we can add more computing power to produce better images. Our impression is that after optimizing the parameters to a certain point using a fixed resolution, the human eye will not spot any significant differences going further with even more computing power.

The most important parameters for the quality of the resulting image are in our opinion the SPP and shadow rays. The increase in shadow rays makes the smoothness in shadows better, but increase in SPP makes both the shadows and the well lit areas more smooth. The depth does however not have that big of an impact on the resulting image. This is mainly due to the Russian roulette terminating most of the rays in only a few steps.

The computing aspect of ray tracing is always a major issue. In table 1 the rendering times are pretty reasonable, but when increasing the parameters the times goes up significantly. For the final render the computing time was 311 min. This time could however in several way be decreased by for example multi threading, running calculations on the graphics card or photon mapping.

By using multi threading, the computations could have been done in parallel by for example calculating one pixel at the time on each thread. This however could be improved even more by using the threads on the graphics card. Since the graphics card is better at heavy calculations, this together with multi thread rendering could have decreased the rendering time for the final image by a significant amount.

Another way to improve the ray tracer is photon mapping [9]. This technique uses photon emission from the light source to estimate the overall light in the room, create caustics and reduce the amount of shadow rays needed by using shadow photons. These perks would not only have decreased the time it takes to render the image for our ray tracer, but also created realistic caustics for the specular objects.

Using a Monte Carlo integration for the ray tracer will introduce noise into the scene due to the randomness of Monte Carlo. For our scene, this was not too much of a trouble since we used a quite high absorption coefficient for all diffuse materials. This caused the noise to not appear as strong, especially when using a higher amount of SPP. However, this resulted in less color bleeding between materials which in some cases might be desirable, but for our case we prioritized less noise.

What is missing from our scene to make it look more realistic is textures. Humans are not used to perfectly flat surfaces, which makes them look unrealistic. The Oren-Nayar reflectors makes it look a bit better but it still does not do the full job. But except for missing textures and some physical phenomenon like caustics, we believe that our scene is a good representation of how lighting behaves in real world and we think that the Monte Carlo ray tracing routine is a good solution for rendering photo realistic images.

References

[1] T. Whitted, *An Improved Illumination Model for Shaded Display*
1980 AMC

[2] H. Wang, W. Zhang, A. Dong, *Measurement and modeling of Bidirectional Reflectance Distribution Function (BRDF) on material surface*, Elsevier

[3] Mark Eric Dieckmann, *TNCG15: Advanced Global Illumination and Rendering, Lecture 1 (Introduction)*, MIT group, ITN, September 3, 2019

[4] Mark Eric Dieckmann, *TNCG15: Advanced Global Illumination and Rendering, Lecture 8 (Idea of MC scheme)*, MIT group, ITN, September 17, 2019

[5] Curtis Mobley, *Lambertian BRDFs*, hämtad 2019-10-25
http://www.oceanopticsbook.info/view/surfaces/lambertian_brdfs

[6] Askiitians, *Critical Angle and Total Internal Reflection*, hämtad 2019-10-26
https://www.askiitians.com/iit-jee-ray-optics/critical-angle-total-internal-reflection/

[7] L. Wolff, S. Nayar, M. Oren, *Improved Diffuse Reflection Models for Computer Vision*
1998 Kluwer Academic Publishers

[8] T. Möller, B. Trumbore, *Fast, Minimum Storage Ray-Triangle Intersection*, Journal of Graphics Tools

[9] H. W. Jensen, *Global illumination using photon maps*
Department of Graphical Communication, 96:21–30, 1996