Monte Carlo Ray Tracer

TNCG15 Advanced Global Illumination and Rendering

Adrian Andersson
Marcus Gladh

March 23, 2020

Abstract

This report follows the implementation of a ray tracer using the Monte Carlo scheme for the course Advanced Global Illumination and Rendering at Linköping University. The report addresses how rays from the renderer interacts with primitive objects of different surface properties in a closed scene. A more in depth explanation of as to why the Monte Carlo scheme was implemented will be provided. At the end of the report the results are presented in the form of rendered images and an analytic discussion.

Contents

Chapter 1

Introduction

Photorealistic image synthesis (or rendering) is a complex task and has been a distinct subject within computer graphics ever since the 1970s. At that point in time, the computer had only a fraction of the computational power they have today. Regardless, successful attempts to various degrees have been made over the years. As the computers have gotten significantly more powerful, the approaches to rendering has become more delicate.

The most successful attempts at achieving photorealistic images is when the render has been based on a physical light model, in combination with mathematically realistic material properties. The propagation of physical light is both recursive and infinite in nature, which is impossible for computers to compute since their capabilities are limited. Thus, approximations and simplifications need to be made without sacrificing unrealistic behaviours.

In 1986, Kaiya introduced the rendering equation [12] that can be seen in equation (1.1). The equation follows the three principles of optics, principle of global illumination, the principle of equivalence (reflected light is equivalent to emitted light) and the principle of direction (reflected light and scattered light have a direction). This means that for every object in a closed scene should contribute illumination to every other object. Secondly, there should be no distinction between illumination emitted from a light source and illumination reflected from a surface and lastly, the illumination coming from a surface most correlate with the illumination that the surface receives. From this equation almost all global illuminations are based on.

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

Before the introduction of the rendering equation, in 1980, Whitted published a paper of a proposed rendering method using ray tracing [12]. The idea was to be able to simulate complex reflections and refractions. This was something that had never been done as previous models was only capable of simulating diffuse and specular surfaces using the local Phong illumination model. This was done by shooting new rays from the intersection points from the rays traced from the camera. These rays could be of different types, either

they could be reflective rays, refractive rays or shadow rays. The type of the new rays depended on the surface properties of the surface intersected. The advantage of this method was that metal, glass and different diffuse surfaces together with shadows from multiple light sources could be rendered. The method could however not simulate complex light characteristics such as color bleeding and caustics.

With the introduction of the rendering equation in 86', Kaiya proposed an idea in the same paper [6] to extend the Whitted ray tracing method by approximating the integral of the rendering equation using Monte Carlo estimation. By using Monte Carlo estimation, noise will be introduced as the variance increases from the function that is being integrated as samples can be zero. There are techniques that can help to reduce the variance and thus the noise, one being some type of importance sampling. Consider a point on a surface χ, observed by the camera from a direction Ψ_c. The total amount of radiance the point receives comes from considering all incoming directions Ψ_i on the hemisphere around the point. By taking random directions on the hemisphere, the radiance on the point χ can be estimated. Using this method and tracing the new rays through the scene, surfaces color will contribute to other surfaces and thus color bleeding can be achieved. It is this method that is implemented in this report.

A few years after Whitted's method, researchers at Cornell University published a paper in 1984 [11] on a refined method of radiosity. Previously the radiosity method had been used in the engineering field of heat transfer. The refined radiosity method considers all surfaces in the scene to be diffuse Lambertian surfaces. The diffuse surfaces are then divided into smaller patches, which the total amount of light transferred between all patches in the scene can then represented as system of linear equations. Due to the amount of computations needed to solve the system, the system is rarely computed but instead solved using iterative methods. The Lambertian reflectance property of being independent of the observers angle of view, this means that the scene does not need to be rendered again when changing the cameras position in the scene. This feature makes the method useful in games where the light map can be pre-rendered offline and applied to the real time rendering of the game environment. The game engine Enlighten used for *Battlefield 3* and *Need for Speed: The Run*, utilized this technique [4].

Radiosity is a physically accurate model in the sense that it follows the first two principles the rendering equation is based on. The method is however, not used for general purposes as it requires the scene to only contain Lambertian surfaces. This is not really applicable for complex scenes but by combining the method with ray tracing to make a two pass rendering method, a physically accurate model that can handle all types of surfaces can be made. The two pass method uses radiosity to render all direction independent surfaces i.e. the diffuse surfaces and let ray tracing handle all other surfaces, such as specular and transparent surfaces.

The two pass rendering method of using radiosity in combination with ray tracing, extended with Monte Carlo, was the most successful approach of rendering images till 96' when Wann Jensen proposed a new method [5]. His proposal was to replace radiosity with photon mapping. In photon mapping, pockets of energy (flux) are emitted from the light sources and let them intersect with the the surfaces in the scene . When a photon strikes a surface, the information in stored in an arbitrary data structure. In Wann

Jensen's paper he chose a KD-tree [5]. The data about the irradiance distribution in the scene, which the photon map provides, can be used to optimize the rendering processes by the Monte Carlo ray tracer. For example, the photon map can be used to compute optimized sampling directions by determining where accurate computations in the scenes are requires and which areas only needs to be approximated. The photon map can also eliminate unnecessary calculations of shadow rays in the scene by only calculating shadow rays on surfaces that are occluded from the light source.

To determine the accuracy of rendering methods using global illumination, Cornell University introduced a test in 1984 called the Cornell Box [11, 7]. The Cornell box consists of five visible sides, where two of the walls are colored green and red. An area light source is placed at the center of the ceiling, spreading light on the objects placed in the scene. Preferably the objects are of different shapes and with different surface properties. One geometry that occurs very often are spheres. This is because spheres can easily be described mathematically and solving intersections for an implicit surfaces is most of the time faster than with other geometry type. An example of a Cornell Box can be seen in Figure 1.1, from Wann Jensen's paper [5]

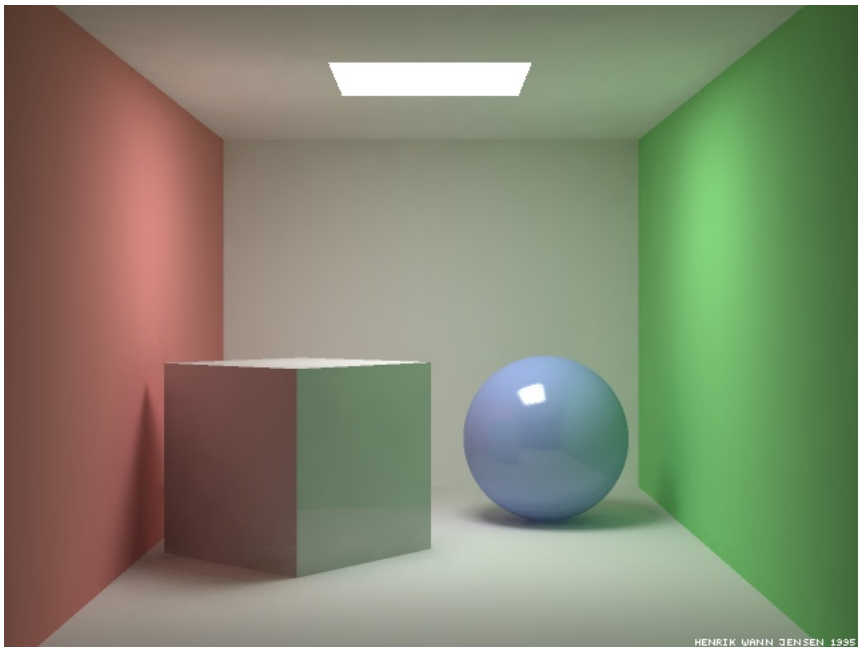


Figure 1.1: Illustration of a Cornell Box by Wann Jensen from his paper Global Illumination using Photon Mapping

Chapter 2

Background

The implementation of the renderer in this report is based on the Monte Carlo method of approximating the integral of the rendering equation.

2.1 Ray-Object Intersection

In a 3D coordinate system, a line can be defined by a point $P_1(x_1, y_1, z_1)$ and the direction vector r_d. This means that the position vector r of any point $P(x, y, z)$ of the line can be described using Equation (2.1):

$$r = r_1 + s \cdot r_d \tag{2.1}$$

Where s is the parameter defining the length of the line and r_1 is the origin. The representation of a ray used in the implementation can be seen in Table 2.1.

Table 2.1: A representation of a ray in the implementation.

Origin	r_0
Direction	r_d
Radiance	L

The radiance variable L contains the information about the energy of they ray, which consists of the colour and intensity. For the implementation the assumption was made that the radiance is preserved along its path.

2.1.1 Ray-Plane Intersection

To testing the intersection of a ray with triangle in the scene, the Möller-Trumbore algorithm was used. The algorithm was introduced by Möller and Trumbore in 1997 [10]

and is to this day considered a fast algorithm and is often used for benchmarks to compare performances of other algorithms. The Möller-Trumbore algorithm takes advantage of the parameterization of an intersection point P in terms of barycentric coordinates. This mean that the intersection point P on a triangle with vertices A, B and C can be computed using Equation (2.2):

$$P = uA + vB + bC \tag{2.2}$$

Where u, v and w are the barycentric coordinates. Both u and v can be obtained using the coordinates of the vertices and in relationship to the incoming ray vector. From the Equation 2.2 and the condition of $u + v + w = 1$, when normalized, w can be obtained. Another important property of the barycentric coordinates is that all the coordinates must follow the properties of $0 \leq u, v, w \leq 1$ and thus if any of the coordinates are greater than one or less than zero, the intersection point is outside of the triangle. Proof of concept and a thorough explanation is presented in a article by *Scratch a Pixel* [9]

2.1.2 Ray-Sphere Intersection

The idea behind solving a ray-sphere intersection test is that spheres, much like lines can be defined in an algebraic format as expressed in Equation (2.3).

$$x^2 + y^2 + z^2 = R^2 \tag{2.3}$$

Where x, y and z are the coordinates of the spheres center and R is the radius. If x, y and z are considered as points, they can be rewritten in the form of a vector P. Substituting P with the equation for a line, the implicit function for a sphere can be obtained, seen in Equation (2.4).

$$|r_1 + s \cdot r_d| - R^2 = 0 \tag{2.4}$$

Developing the implicit function in the form of a quadratic function $f(s) = ax^2 + bx + c$, where $a = r_d^2$, $b = 2r_1 r_d$ and $c = r_d^2 - R^2$, the intersection points on the sphere can be solved. This is done for $f(s) = 0$ when s takes value, using the Equations (2.5)

$$s = \frac{-b \pm \sqrt{\Delta}}{2a} \tag{2.5a}$$

$$\Delta = b^2 - 4ac \tag{2.5b}$$

The Greek letter Δ is called the discriminant and can be used as a simple condition to check the number of solutions given in Equation (2.5a). If Δ is larger than zero, the equation has two solutions. If Δ is equal to zero, the equation has one solution. Lastly, if the Δ is less than zero, there is no solution and there is no intersection with the sphere. An illustration of this can be seen in Figure 2.1.

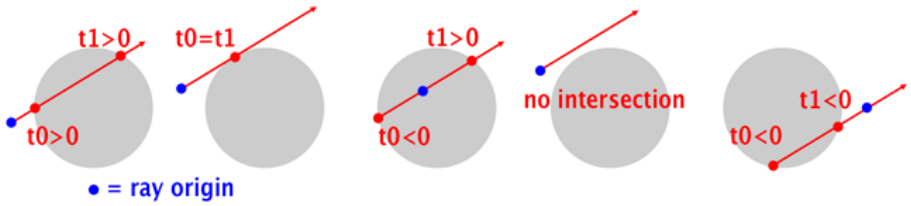


Figure 2.1: Cases for solving the ray-sphere intersection analytically - Scratchapixel, Ray-Sphere intersection (2019-10-24)

The normal of the surface at the intersection point can be obtained through taking the normalized vector between the intersection point and center of the sphere. The normal is crucial when determining the direction of the reflected or refracted ray's direction.

2.2 Bidirectional Reflectance Distribution Function

The bidirectional reflectance distribution function (BRDF) is a function of four real variables that determines how a surface reflects light, depending on the direction of the incoming light relative to the normal of the surface. The BRDFs implemented in the renderer are perfect reflectors (transmissive and mirror surfaces), Oren-Nayar reflector and Lambertian reflector. For the perfect reflectors BRDFs Schlick's approximation is used to approximate the Fresnel Equation. The diffuse reflectors were combined into one reflectance model as the Oren-Nayar reflectance model is the generalization of the Lambertian reflectance model. This is because the Oren-Nayar surfaces can be seen as v-shaped micro-facets that are Lambertian reflectors, covering the surface. The roughness σ in the Oren-Nayar model refers to the probability function for the distribution between the micro-facets, i.e. the standard deviation. A roughness of value zero would mean that the surface is a Lambertian reflector, emitting the same amount of light regardless of the observer's angle of view. The implemented BRDF for diffuse surfaces can be seen in Equations (2.6) to (2.7d):

$$L_r = \frac{\rho}{\pi} \cos\theta_i (A + (B \cdot max[0, \cos\phi_i - \phi_r] \sin\alpha \tan\beta)) E_0 \tag{2.6}$$

$$A = 1 - 0.5 \cdot \frac{\sigma^2}{\sigma^2 + 0.33} \tag{2.7a}$$

$$B = 0.45 \cdot \frac{\sigma^2}{\sigma^2 + 0.09} \tag{2.7b}$$

$$\alpha = max(\theta_i, \theta_r) \tag{2.7c}$$

$$\beta = min(\theta_i, \theta_r) \tag{2.7d}$$

Where θ_i and ϕ_i are the azimuth and zenith angles of the incoming ray and θ_r and ϕ_r are the angles of the outgoing ray, all relative to the normal of the surface at the intersection point. The coefficient ρ is the surface albedo, which is a measure of how much diffuse

reflection out of the total irradiance the point receives.

2.3 Monte Carlo Ray Tracing

In the physical world, light scatters in all directions from diffuse surfaces. To accurately model this, an infinite amount of directions would need to be sampled around the hemisphere on each intersection with a diffuse object. This is not the case for purely specular and transparent surfaces since the outgoing ray can be calculated according to the laws of perfect reflection and Snell's law. A Monte Carlo estimator is used in order to receive the expected value of the incoming radiance on a diffuse surface since there is no analytical way to solve this.

The Monte Carlo method is based on the idea of selecting random samples to estimate an average. The average we seek is the incoming light at our intersection point of a diffuse reflector, i.e. of the types Lambertian or Oren-Nayar. This is a recursive relation because the sample directions might intersect with another diffuse reflector. Figure 2.2 demonstrates an example of this.

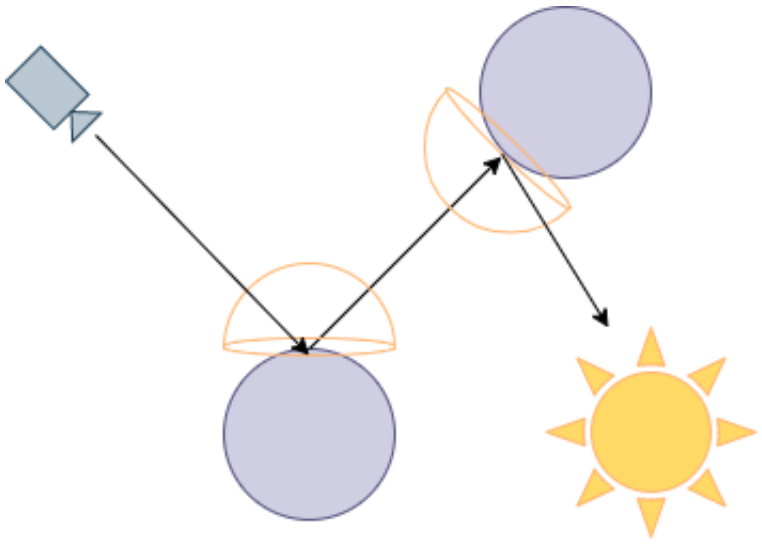


Figure 2.2: An example showing the recursive nature of diffuse surfaces sending off sampling rays in random directions. Each intersection point also presents the hemisphere from which the sample directions are randomly chosen. Both spheres are diffuse reflectors.

Because random directions are chosen each sample is a stochastic variable and thus the average is called the expected value, this is a mere approximation of the true average [8]. The expected value is seen in equation 2.9. The random nature of the Monte Carlo approximation introduces noise in the image. The error in the form of noise is related to the variance of the random numbers and the error is the standard deviation σ [2]. To combat this more samples are needed for a better approximation, the noise however is only reduced by the square of the total number of samples, the relation can be seen in equation (2.8).

$$\sigma \propto \frac{1}{\sqrt{N}} \tag{2.8}$$

$$E(X) \approx \frac{1}{N} \sum_{n=1}^{N} x_n \tag{2.9}$$

In equations (2.8) and (2.9) the N is the total number of samples. The larger N the more our expected value converges to the true average and the error becomes smaller. The Monte Carlo estimator is unbiased which means that the expected value will converge to the true average value, this is important because the goal is to have a photorealistic image. The estimator is used to replace the integral for incoming light at each intersection with a summation, namely the expected value. Equation (2.10) presents both the analytical solution and the Monte Carlo approximation.

$$L_P = \int_{\Omega} L_{\Omega} \tag{2.10a}$$

$$L_P \approx \frac{1}{N} \sum_{n=1}^{N} L_n \tag{2.10b}$$

Equation (2.10a) demonstrates the analytical solution to which there is no solution. Note here that the L_Ω in equation (2.10a) denotes the incoming radiance and is a simplification of the rendering equation. In equation (2.10b) L_n denotes the incoming radiance in a specific sampling direction. In this implementation however only one sample is used for each diffuse intersection, see figure 2.2, thus $N = 1$. This is due to the fact that by having only one sample per intersection more rays can instead be launched through each pixel and thus yield a similar result, this is called subsampling.

2.3.1 Ray Termination

To be able to solve the integral numerically with the Monte Carlo method a way of stopping the rays from spawning new rays on each intersection with a diffuse surface are necessary. This is called ray termination. A popular but naive way to solve this is the stop the recursive calls after a predetermined depth. This is however biased and is therefore not preferred. A better way to terminate rays is with a method called Russian roulette. Russian roulette is a fitting name since the method i based on drawing a random number in the range $[0, 1]$. A ray is only allowed to spawn a new ray if the random number is greater than a predetermined coefficient α. Thus a ray is terminated on each diffuse surface with a probability of α and is allowed to be reflected with a probability of $1 - \alpha$. This is unbiased since each ray has the same probability of being terminated. On purely specular and transparent surfaces rays are never allowed to terminate, while the ray is always terminated if it intersects with a light source.

Terminating rays will affect the expected value and introducing a bias since on average $\alpha\%$ is terminated. To solve this bias a factor of $\frac{1}{1-\alpha}$ needs to be introduced to equation (2.10b) [2]. The new equation is seen in equation (2.11).

$$L_P \approx \frac{1}{N(1-\alpha)} \sum_{n=1}^{N} L_n \tag{2.11}$$

2.3.2 Sampling

To reduce the noise introduced by the Monte Carlo approximation more samples are needed for each pixel. Each pixel is divided into multiple sub pixels and rays are launched through a random position in each of these sub pixels, see figure 2.3. The rays launched through each pixel then gather radiance in the scene and the final pixel value is computed as the average of the total gathered radiance.

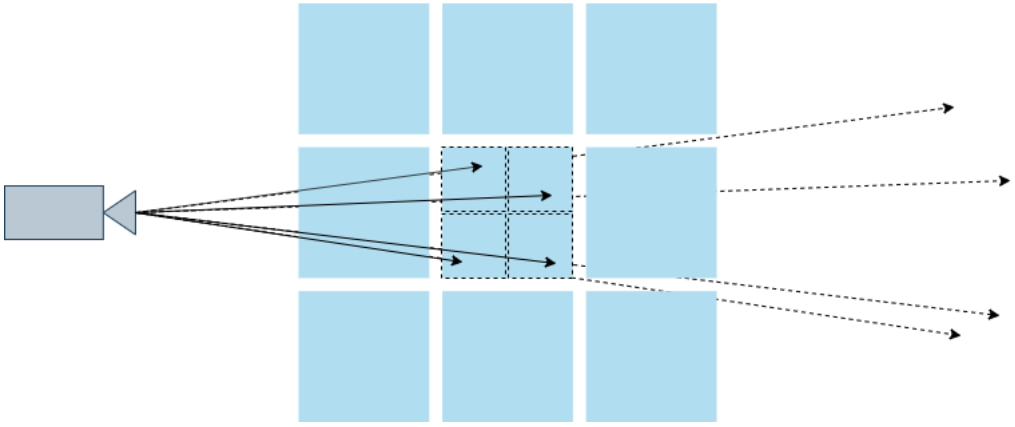


Figure 2.3: Rays shot from the eye point to the camera grid. Each ray hits a random position in each of the sub pixels. The sub pixels are represented by the dotted grid in the middle pixel of the camera grid.

The sub pixels are created so that the random positions does not coincide with each other. Aliasing is also reduced with increasing the number of samples per pixel, hence referred to as SPP [1].

2.4 Light Source & Shadow Rays

At least one light source is needed to illuminate the scene. A area light source is defined as an area from which radiance is emitted. It is assumed that the emitted radiance is constant on all points on the light source's surface. All images in this paper uses an area light source of size 4m^2 and irradiance $1000\text{W}/\pi\text{m}^2$.

At each intersection a visibility check is needed to test if the surface is illuminated or not. To do that we shoot shadow rays from the intersection point to random locations on the area light source. If the shadow rays intersect other objects in the scene before reaching the light source there is no contribution from those to the illumination [1]. Transparent objects

are ignored by the shadow rays since radiance can pass through them. Equation (2.12) demonstrates the area light source's contribution to the illumination on the intersection point [2].

$$L_D(x_N \to -\omega_N - 1) = \frac{AL_0}{M} \sum_{k=1}^{M} f_r(x_N, \omega_{in,k}, -\omega_{N-1}) \frac{\cos \alpha_k \cos \beta_k}{d_k^2} V_k \tag{2.12}$$

Where A is the light source's area, L_0 irradiance of the light source in unit $\mathrm{W\,m^{-2}}$, M the total number of shadow rays, f_r is the surface's BRDF, and α the angle between the light source's normal and the reverse direction of the shadow ray while β is the angle between the surface's normal and the shadow ray's direction. The distance to the light source is denoted d. The factor V_k is the visibility term, it is defined as 1 if the shadow ray is not intersected, and 0 otherwise. If more than one shadow ray is used the resulting shadows will gradually fade away when transitioning into an illuminated area. If multiple light sources are used shadow rays should be launched toward each of those.

Chapter 3

Result

The implementation of the renderer was done in *C++* using the Clang compiler, together with *GLM*, a header only library for vector and matrix mathematics. The computer used to render all images presented in the result used a *AMD Ryzen 7 3700X 3.6 GHz 36MB processor* with *16 GB RAM* running on *Windows 10*.

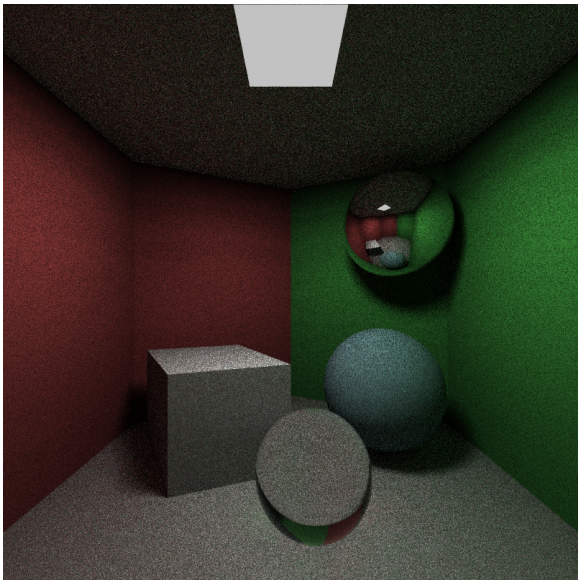
The scene used to test the implemented renderer is a modified Cornell box, with six sides instead of the traditional four. All the walls, floor and ceiling are Lambertian surfaces, with the walls colored in red and green respectively. In the scene three spheres were placed, one transparent, one perfect reflection and one Oren-Nayar with $\sigma = 0.5$. A box was also placed next to the spheres with an Oren-Nayar surface, with $\sigma = 0.3$.

In Figure 3.1 various settings for the number of samples per pixel (SPP) are presented. It is clear that in 3.1a there is a lot of noise present and that the noise is gradually reduced with the increasing of SPP. The light source is located at the same plane as the ceiling and thus the only light that reaches the ceiling is indirect light. On the side of the cube that is facing the red wall, as well as the edges where the ceiling meets the walls, there color bleeding can be seen as a result of indirect light contribution. In table 3.1 the rendering times and other specifics are included.

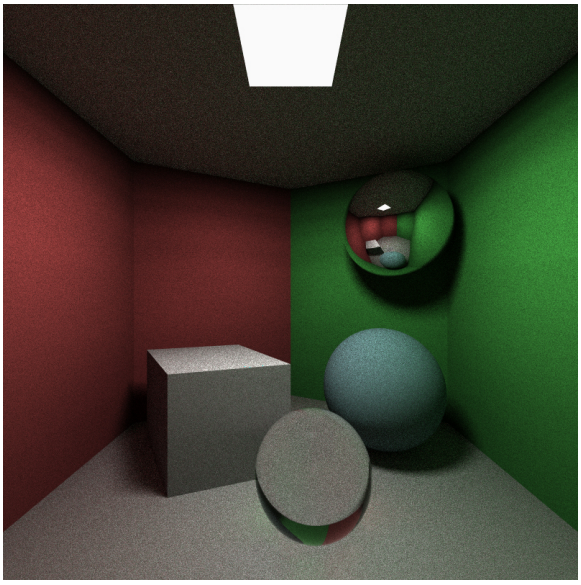
To test the impact of number of shadow rays had on the transition from not occluded to occluded surfaces, three tests were made using the same amount of SPP and resolution. The results can be seen in Figure 3.2.

Table 3.1: Statistics for the produced images.

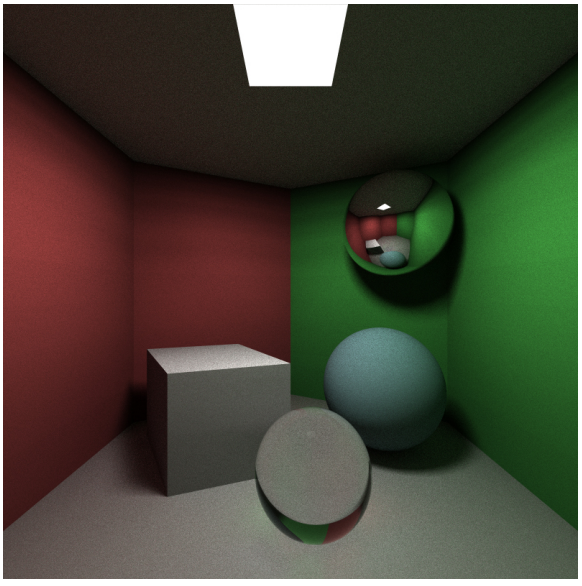
Figure	Resolution	Shadow rays	Samples per pixel	Rendering time (s)
3.1a	800x800	3	4	207
3.1b	800x800	3	16	803
3.1c	800x800	3	64	3190
3.1d	800x800	3	100	5019
3.2a	800x800	1	100	3629
3.2b	800x800	3	100	5019
3.2c	800x800	5	100	6935



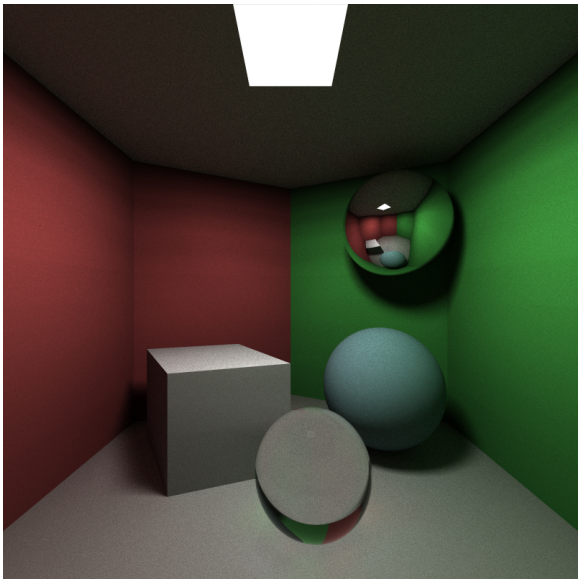
(a) 4 SPP with 3 Shadow rays



(b) 16 SPP with 3 Shadow rays

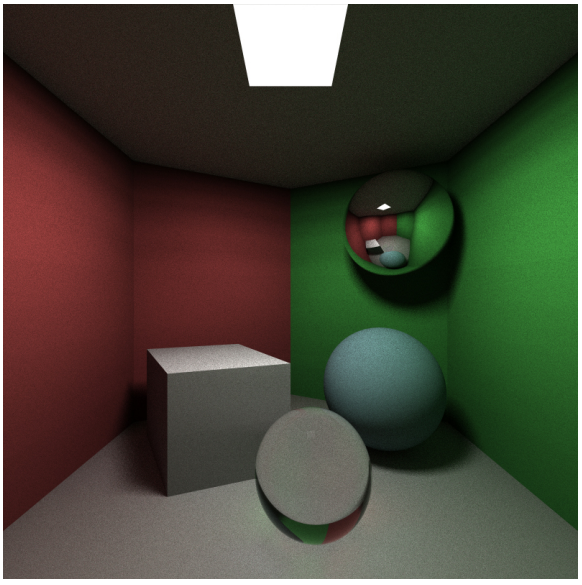


(c) 64 SPP with 3 Shadow rays

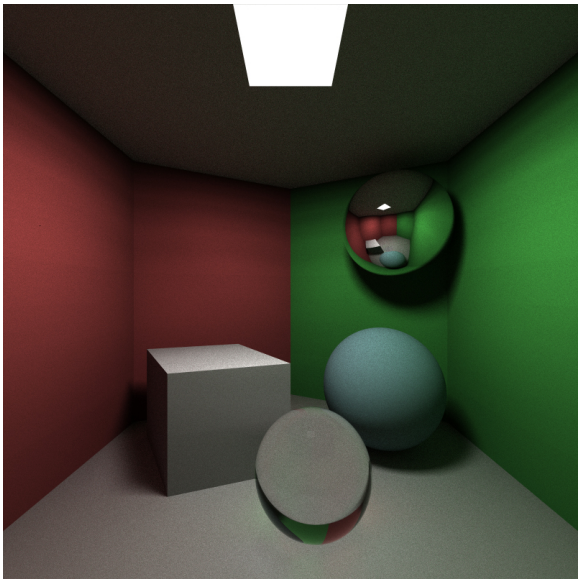


(d) 100 SPP with 3 Shadow rays

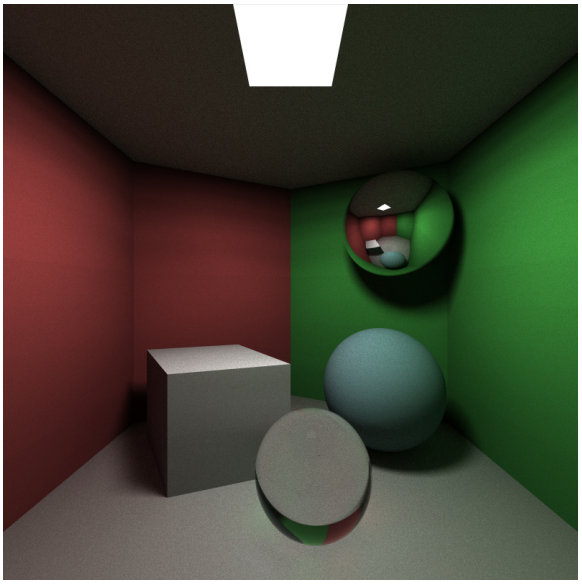
Figure 3.1: Resulting images with different samples per pixel.



(a) 100SPP with 1 shadow ray



(b) 100SPP with 3 shadow rays



(c) 100SPP with 5 shadow rays

Figure 3.2: Rendering of different amounts of shadow rays. All images were rendered with 100 SPP.

Chapter 4

Discussion

As the SPP is increased so is the rendering times. According to the results presented in section 3 a good trade-off seems to be somewhere around 64 SPP. When using more SPPs the noise reduction is only barely noticeable while the rendering times are increased linearly. This is again due to the fact that the noise is reduced by only the square of the number of samples, see equation (2.8). To further expand the scene glossy objects can be modeled using the Monte Carlo approximation. Glossy objects scatters light in a cone around the perfect reflection direction and is thus not that different from a diffuse surface [3]. Caustics are not present in figure 3.1 but can be exhibited if a photon map is used in a two-pass rendering method.

None of rendering times demonstrated in table 3.1 are appropriate for real time usage. However there has been incredible progress on real time ray tracing and it might be reality in a foreseeable future. Surely optimizations could be made to the code, for example an octree can be used to store the triangles and thus reducing the intersection tests from running in $O(N)$ to $O(logN)$ but since the scene is so simple it seemed unnecessary.

From the test conducted with the number of shadow rays, the time required to render the images were significantly far apart compared to the quality of the shadows gained. Depending on the desired quality, a higher number of shadow rays are always preferred but it seems that for general cases, a reasonable amount of shadow rays using this implementation would be around two or three as one does show traces of grain round the edges.

The implementation of the renderer was done from scratch in *C++*. This means that the rendering was done on the *CPU* in contrast to the *GPU* if the implementation was done in *OpenGL*. Since *GPU:s* have more core processors than a *CPU* and are optimized to compute large amounts of mathematical operations in batches, a performance boost in terms of rendering time is to be expected.

Bibliography

[1] Kavita Bala. *CS4620/5620: Lecture 36 Ray Tracing (Shading and Sampling)*. 2012. URL: https://www.cs.cornell.edu/courses/cs4620/2012fa/lectures/36raytracing.pdf.

[2] Mark E. Dieckmann. *TNCG15: Lecture slides*. ITN, 2019.

[3] Philip Pedersen & Tobias Elinder. *Ray tracing with Compute Shader*. URL: http://fileadmin.cs.lth.se/cs/Education/EDAN35/projects/16TobiasPhilip_RayTracer.pdf.

[4] Geometrics. *Enlighten, Game Engine*. 2019. URL: https://www.siliconstudio.co.jp/middleware/enlighten/en/.

[5] Henrik Wann Jensen. *Global Illumination using Photon Maps*. Technical University of Denmark, 1996.

[6] James T. Kajiya. *The Rendering Equation*. Siggraph '86 (Volume 20,Number 4). AMC New York, 1986.

[7] Simon Niedenthal. *Learning from the Cornell Box*. Leonardo page 249-254. Malmö University, 1984.

[8] Scratch a Pixel. "Mathematical Foundations of Monte Carlo Methods". In: (2019). URL: https://www.scratchapixel.com/lessons/mathematics-physics-for-computer-graphics/monte-carlo-methods-mathematical-foundations.

[9] Scratch a Pixel. "Möller-Trumbore Algorithm". In: (2019). URL: https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-rendering-a-triangle/moller-trumbore-ray-triangle-intersection.

[10] Tomas Möller & Ben Trumbore. *Fast, Minimum Storage Ray/Triangle Intersection*. Prosolvia Clarus AB, 1997.

[11] Cornell University. *Modeling the Interaction of Light Between Diffuse Surfaces*. Siggraph '84 (Volume 18,Number 3). AMC New York, 1984.

[12] J. Turner Whitted. *An improved illumination model for shaded display*. J.D. Foley Editor, 1979.