Monte Carlo Ray Tracing - TNCG15

Johnny Elmér - jonel107@student.liu.se
Mithushan Krishnamoorthy - mitkr769@student.liu.se

April 2022

Abstract

This report introduces the rendering equation and describes different methods, such as Whitted ray tracing, radiosity, Monte Carlo and photon mapping, which can and have been used to solve it. The background focuses on the Monte Carlo method and how it is implemented to render an image of a scene containing objects of different materials, such as Lambertian, mirrors and transparent objects. The concept of importance and how different types or ray-object intersections, such as ray-triangle and ray-sphere intersections, work are also discussed. The basic overview of the code structure is also given. Several images are rendered using the Monte Carlo method with differing samples per pixel and show that the method is able to smooth out shadows and create color bleeding effects, which make the images more photorealistic. Some issues still remain however, such as graininess and high computational time, to which photon mapping is given as a possible option for improvement.

Contents

Chapter 1

Introduction

Here the rendering equation, as well as different illumination methods that use said rendering equation are introduced.

1.1 The rendering equation

A rendering equation is an equation which describes how much light is emitted from a point, x, in a specific viewing direction with the help of a function describing the incoming light and a BRDF. The rendering equation is a recursive function, which makes it incredibly computation heavy to calculate for a large amount of recursions, and nearly impossible by hand. Because of this it is impractical to calculate it just as it is and therefore different methods have been developed to solve the equation through, among other things, integral estimations. These methods will be described in sections 1.2.1 through 1.2.4.

1.2 Illumination methods

The concept of ray tracing has been around for centuries, as there was a painter and theorist named Albrecht Dürer[1] that presented the basic principal of ray tracing as early as 1525 in a woodcut print. With the invention of the computer new possibilities were made available and with it came the idea of using ray tracing to produce shaded images on the screen. Arthur Appel was the first to succeed in coming up with techniques that used ray tracing to shade solid objects in a computer and published his findings in the research paper [2] in the year 1968. Appel's solution was to trace rays from the camera and through a scene until it hit the first object it came across. When the ray hit an object a local lighting model was used, for example Phong, to calculate the illumination on that specific point. Appel's approach has with time been adjusted and reworked as computer graphics became more complex and computers able to process far more complicated computations fast.

1.2.1 Whitted ray tracing

For an image to seem photo-realistic it is not enough to use a local illumination model as Appel did, instead a global illumination model is needed. The first global illumination method that could be used for ray tracing recursively was presented by Turner Whitted in [3] in 1980, a whole twelve years after Appel's research was published. The Whitted ray tracing, as it was named, is a recursive ray tracer that allows for rays to be traced from the camera and through the scene like previous methods, but instead

of using a local illumination when a surface is hit, the ray reflects and refracts like realistic light, taking into account the entire scene when the intensity values of each colour is calculated. Before publishing his finding Whitted also produced a short film called *The Compleat Angler*[4] with the help of the recursive ray tracing method he had created.

Whitted's solution for ray tracing, specifically the recursive aspect, was the first one to be able to trace light rays through both reflections and refractions in mirrors and translucent objects respectively. The method was also able to produce anti-aliasing effects through ray tracing, as well as trace shadow-rays to identify whether an object was shaded or not.

1.2.2 Radiosity

Radiosity is, like Whitted ray tracing, a global illumination method. Unlike Whitted ray tracing however this method is not a ray tracing method, but a post processing shortcut to ray tracing. The method takes an already rendered 2D image as input and returns the same image but with radiosity applied to it. It is completely unable to handle specular objects like mirrors or transparent objects like glass. Instead, it is only able to work with objects that are diffuse that only reflect diffuse lighting. The method is good for calculating soft shadows and indirect illumination, or when ray tracing isn't a valid option.

1.2.3 Monte Carlo method

The method that was used in this project was the Monte Carlo method. The Monte Carlo method (hereon called MCM) is used in so called path tracing to simulate global illumination trough recursive ray tracing. The main goal is to create an image which is as realistic as possible by, like with Whitted ray tracing, taking the entire scene into account when illuminating a point in the scene.

Path tracing using the MCM, as mentioned previously, is somewhat similar to a Whitted ray tracer, but the similarities do not stop after simulating global illumination and being recursive. They also share the fact that they send rays from the camera and into the scene, the ability to trace specular reflections and light through translucent objects, but after that they differentiate from each other. The ray tracer is able to to do some things that were previously mentioned in 1.2.2, such as dealing with colour bleeding, soft shadows and diffusely lit surfaces. Unlike the Whitted ray tracer and radiosity however it deals with these calculations by, at each intersection of rays, sampling light from the surfaces of area light sources. As mentioned above, the more samples the more accurate the estimation of the integral, which in turn will produce a less noisy result.

It can also handle reflections from one diffuse surface onto another (diffuse interreflections) and reflections between specular and diffuse surfaces (specular-diffuse surface interreflections).

1.2.4 Photon Mapping

As mentioned in 1.2.3, the lower the number of samples the more noisy the image gets. This noise is not something that is wanted and can be avoided by using the method described by Jensen in [5]. Jensen presented a two-pass method where two photon maps are constructed during the first pass, which handle caustic and global photons respectively, then storing all photons inside of a balanced kd-tree to increase efficiency and reducing the memory required for storing each photon. The second pass takes care of the rendering and uses Monte Carlo ray tracing, similarly to how it is described in 1.2.3.

1.3 Surfaces and objects

There are two different types of object surfaces used in the project: Lambertian reflectors and perfect reflectors. Lambertian reflectors are objects which are "matte" or in other terms diffuse reflectors. The perfect reflectors are, as the name suggests, objects that reflect all of the light that hits it, like for examples mirrors or other shiny, specular objects.

Just like there are different types of surfaces there are also different types of objects: One implicit (spheres) and one polyganol (pyramid). The spheres and pyramid both have lambertian surfaces in the implementation, while one of the walls is a perfect reflector.

Chapter 2

Background

There are three steps required to achieve a global illumination. First is getting measurements and acquiring data, this is done by defining the objects in the scene, specifically their type, location, material property, shape, size, etc. This step also requires defining whether the objects are implicit or polygonal, where the polygonal objects will need to have their line segments defined in some way. The next step is light transportation, where information from the scene and the light sources are used to calculate the light distribution in the scene which are given in radiometric values in the image plane. The third and final step is creating the final image with the help of photometry, where each pixel is coloured with the help of the radiometric values calculated in step two.

2.1 The rendering equation

There are several versions of the rendering equation that can be used. One version is the rendering equation presented by Kajiya in [6], as seen in equation 2.1, although this version was an extension of the Whitted ray tracing, using Monte Carlo integration.

$$L(x \to \theta) = L_e(x \to \theta) + \int_{\Omega_x} f_r(x, \Phi \to \theta) L(x \leftarrow \Phi) cos(N_x, \Phi) d\omega_\Phi \tag{2.1}$$

Another way of writing it (see eq. 2.2), as well as the related BRDF (see eq. 2.3 & 2.4) , is how it is introduced in the introductory lecture of the course (TNCG15) [7].

$$L(x \leftarrow \omega) = L_e(x \leftarrow \omega) + \int_{\omega_1} f * (x_1, -\omega, \omega_1) L(x_1 \leftarrow \omega_1) d\omega_1 \tag{2.2}$$

$$f * (x_1, -\omega, \omega_1) = f_r * (x_1, -\omega, \omega_1) cos\theta_1 \tag{2.3}$$

$$f_r * (x, \omega_{in}, \omega_{out}) = \frac{dL(x \to \omega_{out})}{dE(x \leftarrow \omega_{in})} \tag{2.4}$$

2.2 Monte Carlo Method

The goal of the MCM, as described in 1.2.3, is to create an image which is as realistic as possible by, like with Whitted ray tracing, taking the entire scene into account when illuminating a point in the scene. The MCM gives a way to estimate the integral value of the integral in equation 2.1 to simplify

the calculation and make it less computation heavy. To do this, random discreet numbers are used to sample the integral function, giving an increasingly accurate solution the higher the sample number becomes. As an example:

$$I = \int f(x)dx \tag{2.5}$$

$$< I >= \frac{1}{N}\sum_{i}^{N}\frac{f(x_i)}{p(x_i)}dx \tag{2.6}$$

The integral seen in 2.3 can be estimated as seen in 2.4 where N is the number of samples which are distributed in accordance with $p(x_i)$ (which is the probability distribution function). When the integral in 2.1's value is estimated the rest of the equation can be solved, now much more efficiently than before.

2.3 Materials

In this section the different types of materials are introduced and explained in terms of how the direction of the reflection and refraction of light is calculated for each type.

2.3.1 Mirror

A mirror is an object which surface is a perfect reflector. For an object to be a perfect reflector it has to reflect each ray hitting its surface perfectly, where the direction of the reflected ray is computed with the help of equation 2.7

$$\hat{R} = \hat{I} - 2(\hat{I} \cdot \hat{N})\hat{N} \tag{2.7}$$

where $\hat{R}$ is the direction of the reflected ray, $\hat{I}$ is the direction of the light ray coming towards the object and $\hat{N}$ is the surface normal of the point of intersection.

2.3.2 Transparent

A transparent object, like for example glass, is an object which refracts most, if not all, of the rays of lights that hits its surface. If an object is supposed to look completely transparent without any imperfections, it is called a perfect refractor. To calculate the angle between the refracted ray and the surface normal, equation 2.8 (Snell's law) is applied

$$\frac{sin\theta_1}{sin\theta_2} = \frac{n_2}{n_1} \tag{2.8}$$

where n_1 and n_2 represent the refractive index for two different materials, for example the air and the glass. θ_1 is the incoming rays angle in relation to the surface normal and θ_2 is the angle we are searching for.

When the angle between the refracted ray and the surface normal (θ_2) has been obtained it can be used inside Schlicks equation (see equations 2.9 through 2.11) to get the reflection coefficient (T), which describes in which direction the refracted ray should travel.

$$R(\theta) = R_0 + (1 - R_0)(1 - cos\theta)^5 \tag{2.9}$$

$$T(\theta) = 1 - R(\theta) \tag{2.10}$$

$$R_0 = ((n_1 - n_2)/(n_1 + n_2))^2 \tag{2.11}$$

2.3.3 Lambertian

Lambertian surfaces are slightly different from mirrors and transparent surfaces in the sense that they are not perfect reflectors or perfect refractors. How much light is reflected from the surface is instead decided with the use of the BRDF, which for Lambertian surfaces is calculated using equation 2.12.

$$f_r = \frac{\rho}{\pi} \tag{2.12}$$

There exists other more complex materials which mimic realistic objects more closely by using a more complex BRDF, but these were not considered for this project.

2.4 Importance

Importance could be describe as a quantity which is emitted from the eye point. Object points in the scene which can be directly seen by the eye point or via reflections receive importance. Importance flow in the opposite direction of the radiance.

2.5 Intersections

In this section the algorithms which calculate the intersection between ray and different type of objects are discussed.

2.5.1 Ray-triangle intersection

The Möller–Trumbore algorithm is used to find the ray -triangle intersection point as shown in Figure 2.1 in this project. This function takes rays and triangles as arguments and return a boolean variable which indicates whether the given ray intersects the given triangle or not.

The advantage of this algorithm is that there is no need to calculate and store the plane in which the triangle lies and it is the fastest ray - triangle intersection algorithm which does not require the plane to be stored.

2.5.2 Ray-sphere intersection

A ray may intersect a sphere once, twice or never, which is different from a triangle which can only be intersected once by a ray as shown in Figure 2.2. First the shortest distance from ray to the centre of the sphere is calculated. If the distance is larger than the radius then there is no intersection, if the

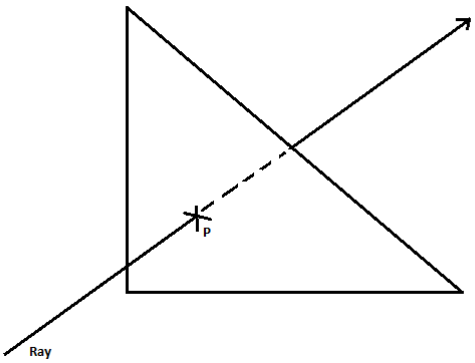

Figure 2.1: Ray-Triangle intersection

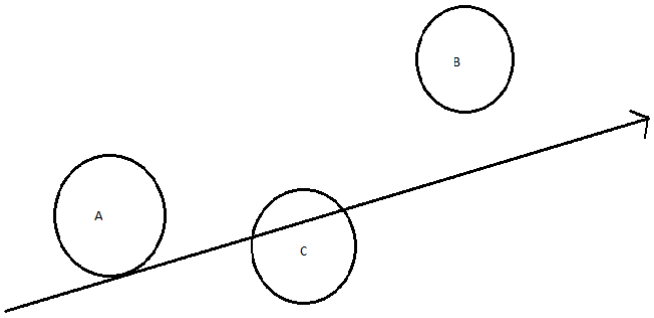

Figure 2.2: Ray-Sphere intersection

distance is equal to the radius then there is only one intersection and if the distance is less than the radius then there are two intersections.

2.6 Code structure

In this section the basic code structure of the Monte Carlo ray tracer is introduced, with a description of each class, what it contains, and what its function is.

2.6.1 Vec3

This class is a vector with three doubles which also contains all the necessary functions to work with a vector. Alias Color, Point3 and Direction are also created here.

Alias Color is used to define the color of a given surface in our 3D scene. Three double precision variables containing the intensities for red, green and blue are used here. Point3 is used to represent vertices, intersections and locations of different objects and Direction is used to represent the direction of a given Ray in the scene.

2.6.2 Ray

A Ray class is crucial in building a ray-tracer. A ray is defined with the equation 2.13, where P is a point in 3D space, A is the starting point of the ray (camera location), b is a direction vector and t is a scalar which changes depending on where in the scene the ray intersects with an object.

$$P(t) = A + tb \tag{2.13}$$

2.6.3 Camera

In the class camera the aspect ratio of the output image is set, as well as height and width of the viewport and the focal length. This class also contains a function which returns a ray given the 2D coordinates of the camera plane.

2.6.4 Material

The class material is used to define the color and properties of a given surface. This class has three subclasses which are, lambertian, metal, and diffuse_light. These are also the three different types of surfaces which make up the whole scene.

2.6.5 Hittable

This is a base class for all the objects in the scene which are supposed to intersect with ray. The classes Triangle and Sphere inherit from Hittable. This class has a virtual function which, depending on if the object is a triangle or a sphere, calls the respective intersection function to determine if the ray intersects with the object.

2.6.6 Triangle

This is a subclass of the class hittable (see 2.6.5). This class contains three Point3 objects which represent the three corner vertices of a triangle. This class also sets the material of the given triangle. The triangle class also contains a function to determine whether a ray intersects with a Triangle with the help of the Möller Trumbore algorithm.

2.6.7 Sphere

The class sphere is also a subclass of class hittable (see 2.6.5). This class defines a sphere object. The centerpoint, radius and the material of the sphere is set in this class. Just like the Triangle class, sphere also contains a function which determines if the ray hits the sphere.

2.6.8 Scene

The scene is a six sided box (as seen in Figure 2.3) created with the help of 24 triangle objects. Each triangle is arranged so that the normal of the triangle points inside towards the centre of the box. Each wall is made of a set of two triangles with same colour. There is one tetrahedron and two spheres inside the box. Triangles intersection point with incoming rays are calculated using Möller–Trumbore ray-triangle intersection algorithm. One of the walls are a perfect reflector and we use a diffuse area light source to illuminate the scene.

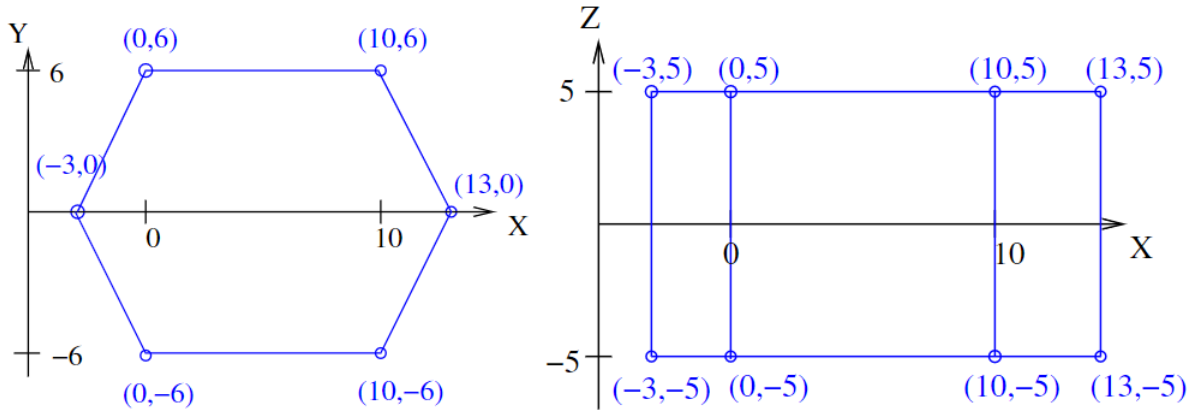

Figure 2.3: A 2D-representation of the scene.

2.6.9 How the image is created

The image is made up of 400 X 225 pixels. Camera plane is a 2D array with 400 x 225 elements. Each element is a Pixel. Through each pixel we send a certain amount of rays. This is also called the samples per pixel (spp). The rays are either reflected or refracted depending on the type of surface.

First test: Let it run until it hits the absolute first triangle and assigns the ray the triangles color.

Method: createImage().

Scene image stored in ColorDbl attribute of Pixels.

Convert data into 2D array of RGB vectors with int values (ranged 0-255).

Chapter 3

Results

In this chapter the results of the implementation are presented with the help of rendered images at different stages and with different samples per pixel as well as different maximum depths of the rays.

Figure 3.1 shows the best looking image rendered by the program using a form of Whitted ray tracing, before the implementation of Monte Carlo.

Figures 3.2 through 3.5 show the rendered images after implementing the Monte Carlo method. Note that some objects have moved position and a mirror has been placed on one of the walls, otherwise the scene is left in large parts unchanged.

As the samples per pixel increases there is less visible pixelation and noise, creating a more cohesive image and helping to smooth out shadows and making them less clearly defined (see Figure 3.6). There is also a slight colour bleeding effect present at the edges where two colours meet, as is seen in Figure 3.7.

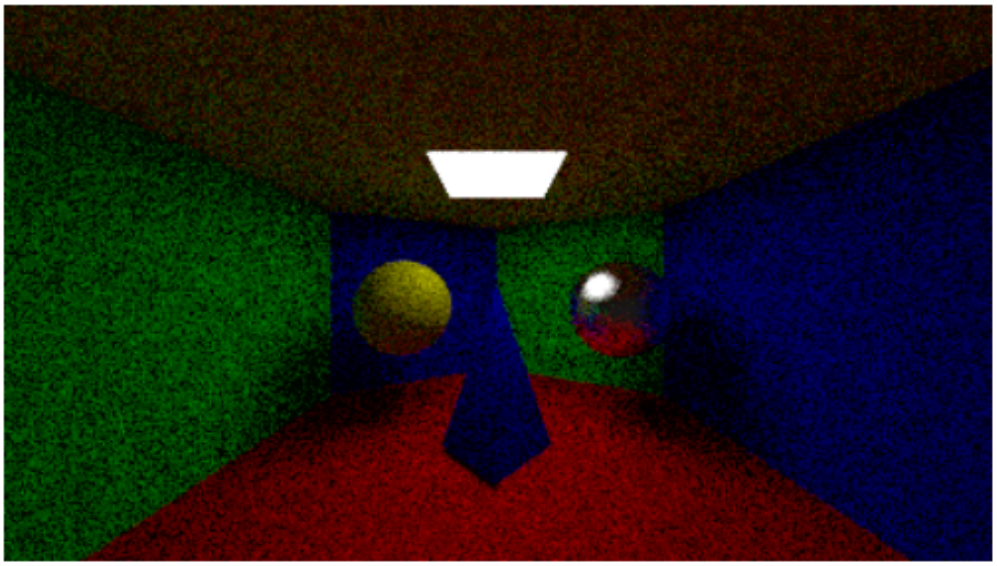

Figure 3.1: Pre Monte Carlo image.

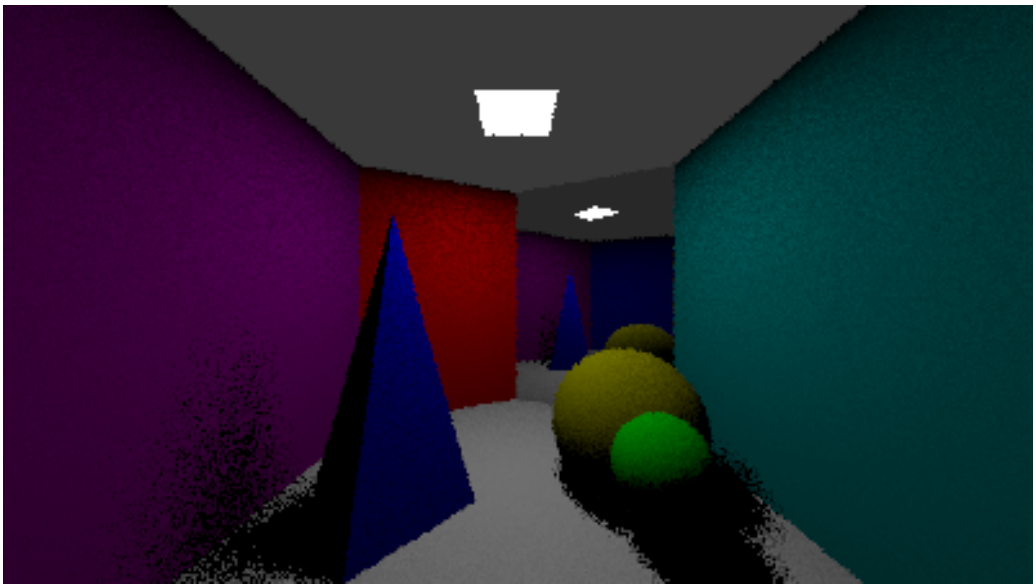

Figure 3.2: Monte Carlo image with 2 spp.

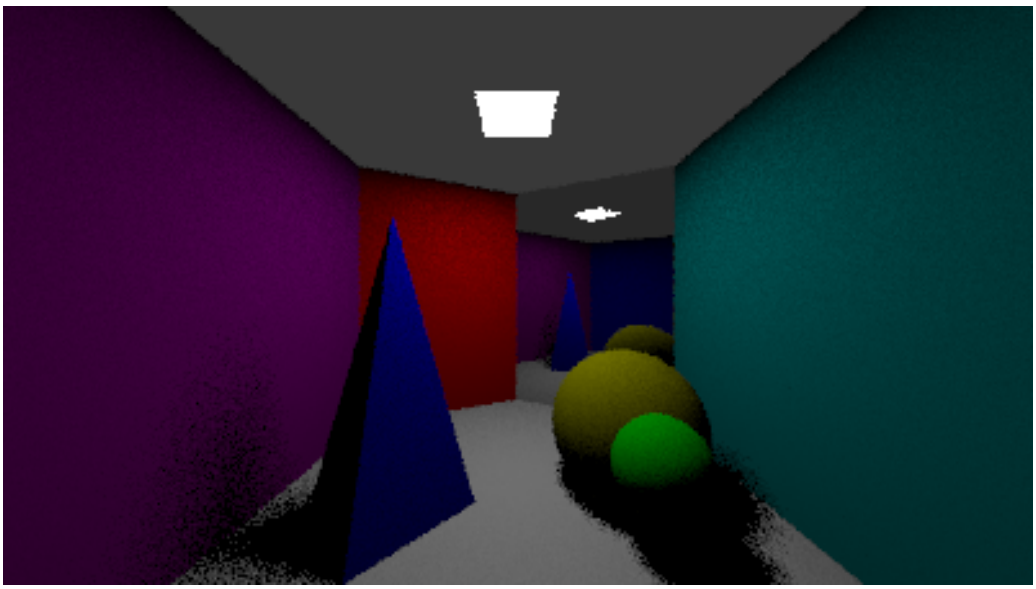

Figure 3.3: Monte Carlo image with 4 spp.

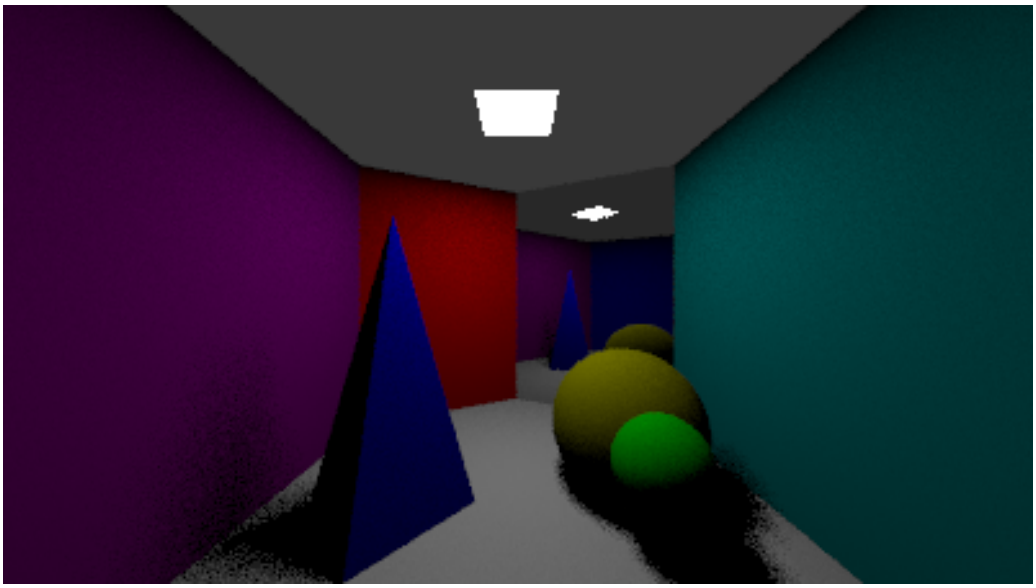

Figure 3.4: Monte Carlo image with 8 spp.

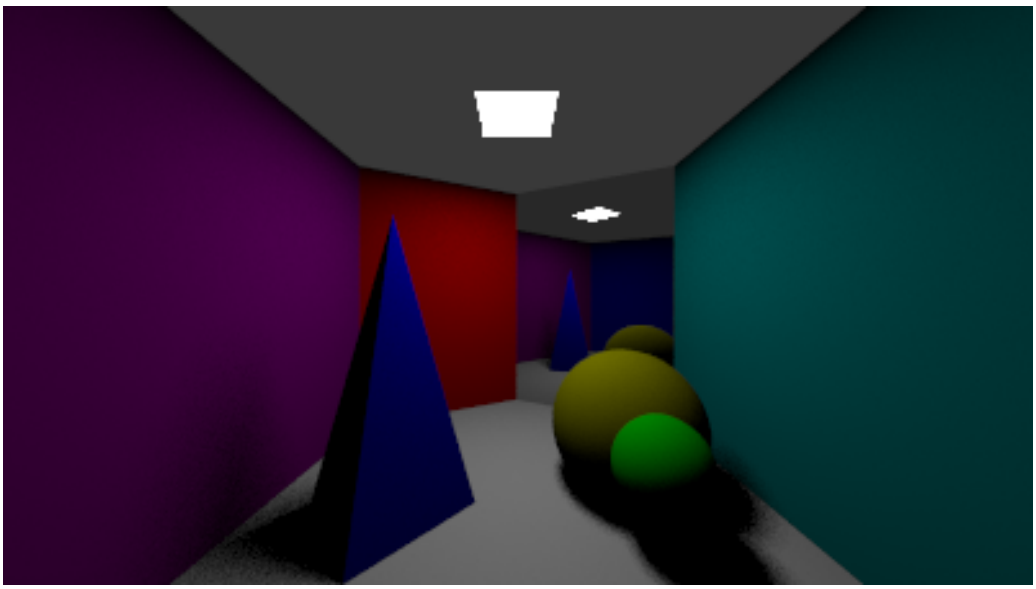

Figure 3.5: Monte Carlo image with 64 spp.

Figure 3.6: A closeup of the soft shadow on the purple wall.

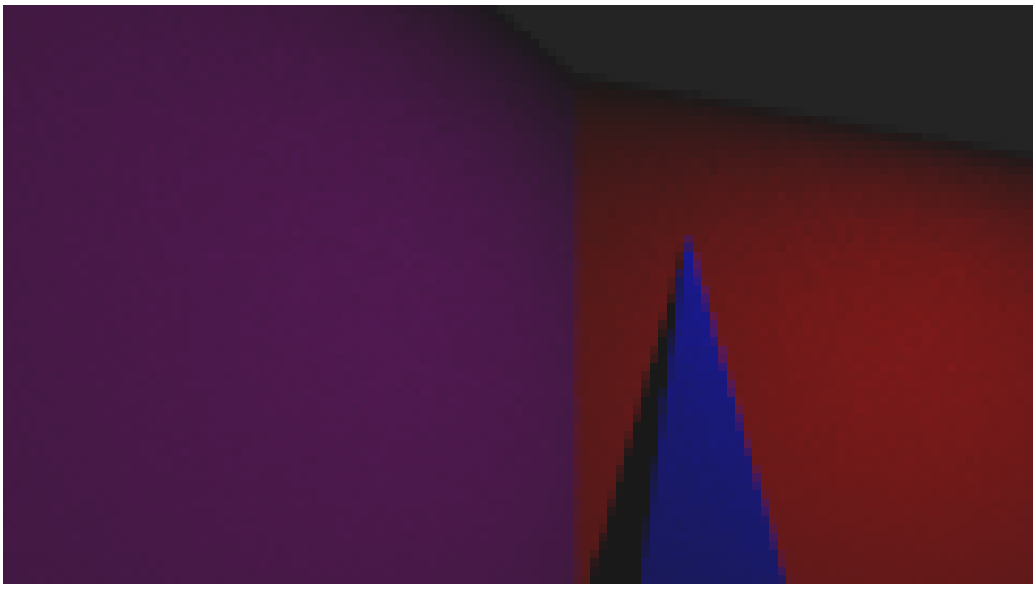

Figure 3.7: A closeup of the colour bleeding effect between the purple and red walls.

Chapter 4

Discussion

In this chapter we discuss the results that were presented in 3.

4.1 Before Monte Carlo

As seen in Figure 3.1 the first simplified ray tracer implementation was able to produce fairly good looking images with appropriate colour bleeding as well as somewhat reflective objects (one of the spheres). Both spheres shadowed the surroundings and so did the pyramid, even if its shadow was harder to see because it was placed right underneath the lamp. All walls, the roof, floor and objects are acceptably lit.

4.2 After Monte Carlo

The Monte Carlo method is able to take care of hard shadow problems and the lack of color bleeding, which are the two main problems in the Whitted ray tracing. The soft shadow effect is achieved by sending the shadow ray to a random point of the light source which make sure that some part of the shadows are fully covered and the edges are soft. A photorealistic image usually has the color bleeding effect due to the indirect illumination from surfaces with other colours. The color bleeding effect is achieved with the help of the Monte Carlo integration.

The Monte Carlo method is very good at creating photorealistic images of a scene but it has some disadvantages. One of the disadvantages is that it requires a lot of computational power and the second one is that the images tend to be noisy. More sample per pixel improves the image quality but it also consume more computational time.

4.3 Possible improvements

Although the results obtained are good and realistic to a certain extent, there is room for improvement. As touched on above, one clear issue is that the images are still showing a slight graininess and that the computational power is rather high. One solution to both these issues would be to implement photon mapping.

Bibliography

[1] Georg R. Hofmann. Who invented ray tracing? https://link.springer.com/article/10.1007%2FBF01911003, 1990.

[2] Arthur Appel. Some techniques for shading machine renderings of solids. http://graphics.stanford.edu/courses/Appel.pdf, 1968.

[3] J. D. Foley and Turner Whitted. An improved illumination model for shaded display. https://www.cs.drexel.edu/~david/Classes/Papers/p343-whitted.pdf, 1980.

[4] Turner Whitted. The compleat angler. https://archive.org/details/thecompleatangler1978, 1978.

[5] H. W. Jensen. Global illumination using photon maps. *Rendering Techniques '96 (Proceedings of the Seventh Eurographics Workshop on Rendering)*, pages 21–30, 1996.

[6] J. T. Kajiya. The rendering equation. *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, 2:143–150.

[7] M. E. Dieckmann. TNCG15: Advanced global illumination and rendering, lecture 1 (introduction). https://liuonline.sharepoint.com/:b:/r/sites/Lisam_TNCG15_2021HT_BL/CourseDocuments/Lecture01.pdf?csf=1&web=1&e=3rpbNa, Aug. 30, 2021.