Monte Carlo Ray Tracing, TNCG15

Jakob Gunnarsson
Nicholas Frederiksen

February 2, 2020

ABSTRACT

This report describes methods for rendering images with global illumination. It also contains descriptions of the renderer, Monte Carlo ray tracer, that was developed for this specific project. Moreover, images that have been rendered in this project are presented. The report also concludes with a discussion about results and some known issues with the implantation.

Contents

Chapter 1

Introduction

There has been a desire since the beginning of computer graphics to develop methods that produce images that are photorealistic. A lot of considerations are needed when calculating the light in order to get a good representation of a physical scene.

Previous shading techniques have used a local illumination model in 3D computer graphics that measures how light from a light source is reflected on an object. The limitation of this model is that the direct emission from the light sources are the only component that influences the light in the scene. The necessities for such restrictions were understandable, the goal was to make realistic results within a reasonable computation time.

However, a global illumination model can consider indirect light as well. Meaning not only light from emission sources but also reflected and refracted light from surrounding objects. With this illumination model, it becomes possible to simulate phenomena like color bleeding, soft shadows and caustics, adding further realism to rendered images. It is safe to say that global illumination algorithms are needed for achieving photorealism. The two most common algorithms/methods are *Radiosity*, first introduced to computer graphics in 1984 [1] and *Ray tracing*, first introduced to computer graphics in 1968 [2]. However, many improvements and further developments have been made since then. Almost all ray tracing algorithms used today are based on the work of Turner Whitted.

Something these ray tracing algorithms all have in common is that their final result is directly dependant on the density of rays sent out through the scene. To avoid noise in the rendered image, the algorithm requires multiple rays per pixel. For this reason it was impractical to use since it required long computation time. It was not until major improvements in the CPUs and GPUs were achieved, when ray tracing techniques could be used to gain high quality rendered images that promised photorealism through an extension named the Monte Carlo ray tracing method. More on Monte Carlo ray tracing is presented in chapter 2.

In this project a Monte Carlo ray tracer has been implemented. It renders polygonal objects with Lambertian reflectance models. The report will first describe different methods used when rendering images with global illumination in the Background and Related work chapter, followed by a chapter on what methods and structures were used to implement a Monte Carlo ray tracer, then a presentation of the resulting images and a discussion.

Chapter 2

Background and Related Work

This chapter will present different approaches for global illumination algorithms. A distinction can be made between two separate models for global illumination; the view-independent and the view-dependent. The models are used to calculate different components of the indirect light arriving at a point on a surface.

The main difference between the two models is that a view-independent illumination model does not rely on where the camera or eye point is located in the scene, taking into account directional independent indirect light, and does not need to be recomputed when moving the viewport.

Moreover, a view-dependent illumination model, taking into account the directional dependent indirect light, is completely dependent on where the camera is placed in the scene (location and direction) and will need to be recomputed as the viewport moves.

Both of the models excels in computing their own contribution to full global illumination and are sometimes preferably used in combination in what is usually refered to as a *Two-pass solution*.

Radiosity and *Photon Mapping* are two common methods for computing the view-independent part of the indirect light.

When it comes to computing the view-dependent part of the indirect light of global illumination, two common methods are *Whitted Ray Tracing* and *Monte-Carlo Ray Tracing* (Distributed Ray Tracing).

2.1 Radiosity

As mentioned earlier, the *Radiosity* method was presented the article "Modeling the Interaction of Light Between Diffuse Surfaces" by Goral et al. in 1984 [1]. Radiosity is a method based on heat transfer (Thermal Dynamics), and was modified for applications in the field of computer graphics. An advantage of Radiosity is that it can simulate finite area light sources and diffuse surface reflections in a scene, meaning that it can provide realistic effects that are otherwise missing in simpler global illumination methods, such as color bleeding and soft shadowing.

The main concept behind Radiosity is to divide the scene into a finite number of patches, then measure for each patch the cumulative Radiosity from all other patches in the scene arriving at that patch, plus the self-illumination patches. This is classified as an iterative algorithm which initially only has self-illumination on the patches corresponding to the light source. Furthermore, every iteration will have the number of patches emitting radiance to all other patches increased, depending on the patch's relative reflective capabilities and the form factor between any patch pair. Thus the algorithm will converge to a true solution as the number of iterations increase [8].

To summarize, the Radiosity method is composed of several passes. The first pass illuminates the re-

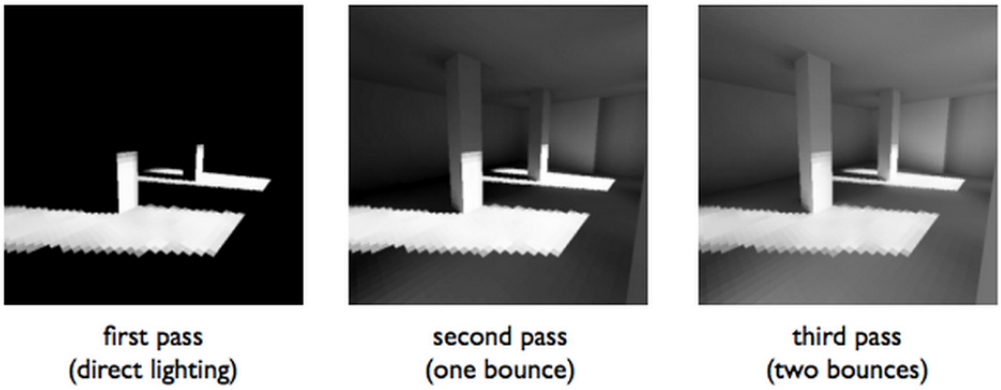


Figure 2.1: Radiosity progress

flections of directly affected surfaces by a light source. Further, the second pass will bring a fraction of the light previously illuminated to visible surfaces around its source. The rendered image normalizes more for each run, and becomes more realistic, see Fig 2.1.

2.2 Photon Mapping

Jensen introduced photon mapping in the 1996 article "World Illumination using Photon Maps" [9]. It is a two-pass approach to global illumination that takes into consideration caustic effects and diffuse reflections between surfaces, see Fig 2.2. It is a method that can quickly simulate realistic effects such as caustics and color bleeding, effects that are often lacking in simpler global illumination algorithms. It was once deemed to be an efficient alternative to Monte Carlo ray tracing. However, nowadays there exist sophisticated GPUs that can perform ray tracing more efficient then Jensen could ever imagine.

Below is a short overview of the two passes in photon mapping. They are as follows:

Pass 1: Light emission and photon scattering. The first pass generates the photons and launches them into the scene from the light source. Photons spread flux, and more photons are emitted by a light source with higher intensity. Every photon will have a randomly chosen direction based on what kind of light source it was released from. Then the photons scatter throughout the scene, and collide with various objects/surfaces. When a photon hits an object, it can be one of three things: *transmitted*, *absorbed* or *reflected*. This first pass constructs two separate photon maps; the first map is a global map representing photons which have been traced through a number of diffuse reflections before encountering a diffuse surface. The second map is a caustic map representing photons which have been refracted or reflected before encountering a diffuse surface [9].

Pass 2: Radiance estimate and rendering. The second pass renders the scene with the benefit of the information stored in the photon map, using a modified Monte Carlo ray tracer. By having a photon map, detached from the geometry in the scene, it is possible to render very complex scenes with global illumination. Furthermore, next rays are launched from the camera into the scene, and when a ray hits a point $\mathbf{P}$ on a surface, the illumination of nearby photons are collected to calculate the radiance contribution at point $\mathbf{P}$. It is possible to determine which photons contribute light by creating a small hemisphere around point $\mathbf{P}$ and considering all photons surrounded by the sphere along with their indicating direction $\mathbf{d}$ [9].

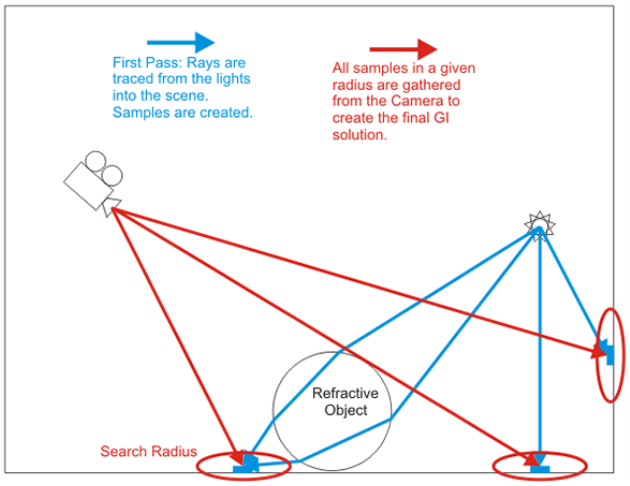


Figure 2.2: Illustration on the way the Photon Map is generated. In the first pass (Blue) rays are traced from each light into the scene which generates light samples across the entire scene. In the second pass all samples in a specified radius are gathered from the camera in order to create the final GI solution.

2.3 Whitted Ray Tracing

The most classical examples of light transport algorithm based on ray-tracing is called *Whitted Ray Tracing*, a method first introduced in 1980 by Turner Whitted.

One significant difference between ray tracing and real-life light is that the rays in ray tracing are tracked backwards. The method essentially generates an image by tracing the path of a light ray throughout a scene. The ray begins at an observer point/an imaginary eye facing a direction, it then goes through a pixel in the image plane and continues it's path towards the scene until it intersects with a surface.

Once this ray has intersected with a surface, a new ray is launched from that point, redirected towards a different direction according to the surface properties, i.e. compute the reflection or refraction direction using the law of reflection or refraction. Then a new ray direction (or two new directions if the surface is transparent) is made, creating a continuous path for the ray. Once the ray intersects with a light source or has intersected a certain amount of times the path is ended. All these rays are then combined to set a color, and apply it to the pixel the ray was shot through. The biggest advantages of this is that there is no need to simulate all light rays coming from each light source, since most of them miss the observer anyway.

Additionally, at each intersection point, to handle surfaces which are shaded by other surfaces in the scene a shadow ray will be cast, directed towards all light sources, as a ray hits a diffuse surface. If the shadow ray ends up intersecting with a nontransparent surface, then the surface point in which the shadow ray came from is deemed to be in shadow and should not be illuminated by the light. This gives the image more realism by simulating shadows, a task which is difficult with other methods. See in Fig2.4 for an illustration on a traced ray path.

At the expense of long computation time, the method can render images converging to photorealism. Because of this, ray tracing is well suited for applications where the rendering can be done in advance, for example in movies, visual effects and still images for adverts. Regarding rendering applications in real time, they may require either some limitations to image quality or modifications to be made in order to meet an acceptable rendering speed.

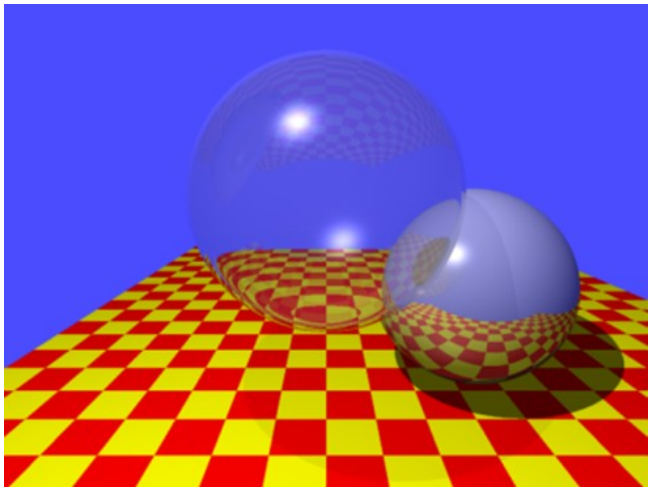


Figure 2.3: Caption

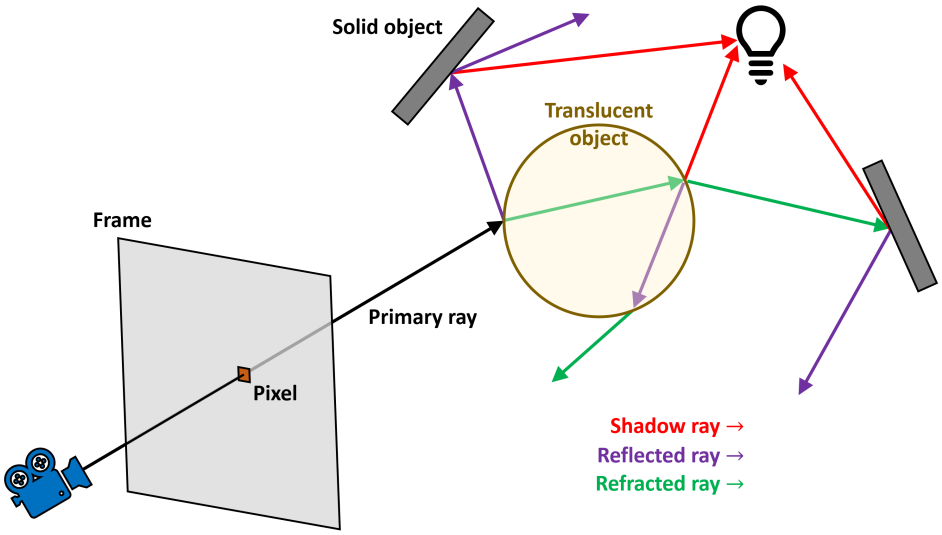


Figure 2.4: Ray tracing example path. Notice that at each intersection a shadow ray is cast towards the light source (red).

2.4 Monte Carlo Ray Tracing

In 1984 an article by Cook et al.[3] was published that proposed an extension to the Whitted ray tracing. The proposed method would enable the simulation of effects such as glossy refractions and reflections, motion blur and depth of field, by distributing rays over several dimensions.

The term Monte Carlo ray tracing or Distributed Ray Tracing as it is sometimes referred to actually mean that Monte Carlo Integration Techniques are applied in order to solve the rendering equation [6]. More on the rendering equation can be read in section 2.4.1.

As opposed to a Whitted ray tracing, the Monte Carlo approach has a number of advantages; it is able to simulate diffuse inter-reflections between surfaces, glossy reflectors, caustics (although this is unusual and inefficient in a pure Monte Carlo ray tracer), area light sources and color bleeding.

The Monte Carlo approach will randomly distribute rays in any direction upon a intersection between a ray and a surface, and the recursion does not stop after a predetermined ray depth but rather after a *Russian Roulette* methods stopping condition [7]. For handling shadows, both hard and soft, caused by surfaces occluding other surfaces in the scene, shadow rays will be randomly cast towards the light source area to determine visibility at a surface point. The derived direct contribution from the light source will be averaged with respect to the number of shadow rays, forming softer shadows.

Moreover, when the Monte Carlo ray tracer in its basic form utilizes only the rendering equation to handle Lambertian reflectors, a property that defines an ideal "matte" or a completely diffuse reflecting surface. When a ray is first sent from the camera/eye point and intersects with a surface it will create a hemisphere around the point of contact facing the direction of the point normal. Rather than integrating over the entire hemisphere as rendering equation proposes, one or multiple random directions on the hemisphere are sampled. With randomized directions, unbiased samples are achieved. However, this may result in noise in the image if not enough rays are sampled. Just like Whitted ray tracing, Monte Carlo ray tracing can be extended to include perfect reflection and refraction laws.

Furthermore, when the ray samples a direction on the hemisphere, it uses the *bidirectional reflectance distribution function* (BRDF) corresponding to that surface, to sample a new direction. The BRDF can be described as the function which holds information about the correlation between in-going and out-going rays [4].

To summarize, Monte Carlo ray tracing differs from Whitted ray tracing in three major ways. They differ in the stopping condition, the directions of the spawned rays and the number of spawned rays. They are otherwise quite similar. In addition, Monte Carlo ray tracing will (if given enough time) produce very realistic images and will be able to deal with any form of surface-light interaction. However, the drawbacks of the algorithm are the long rendering times and the fact that it produces very noisy pictures due to its random but unbiased nature. Although, several well-known methods have been derived over the years for reducing the noise and the rendering time.

2.4.1 Rendering Equation

The Rendering equation is an equation which for each point $\mathbf{x}$ and each direction ω formulates the existent radiance at that surface point and in that direction. Solving this equation is the fundamental goal of global illumination algorithms as it models most light transport phenomena.

In its hemispherical formulation, after some simplifications, the rendering equation can be presented as:

$$L_o(x,\omega) = L_e(x,\omega) + \int_\Omega f_r(x,\omega_i,\omega) L_i(x,\omega_i) d\omega_i \tag{2.1}$$

Equation 2.1 describes the reflected surface color as the sum of the surface's emission and the integral over the BRDF and incoming light. The term $\mathbf{x}$ is the surface position, ω is the direction the light gets reflected towards and ω_i is the *incoming* light direction [7].

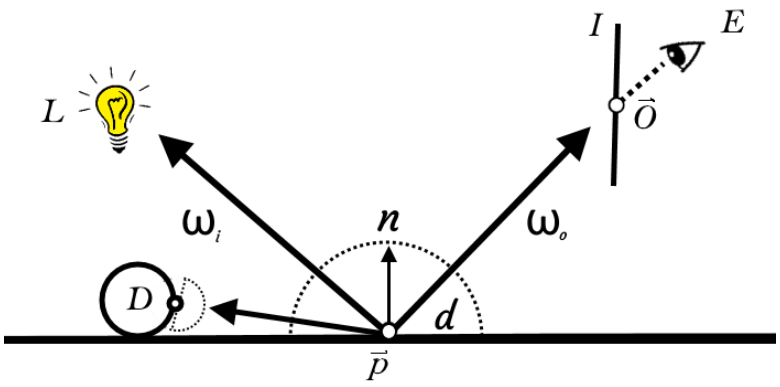
As stated before, the BRDF then describes how the light behaves on a material. More specifically it relates the differential radiance reflected in an existing direction to the differential radiance incident through a differential solid angle, as it is called.

Chapter 3

Method

The following sections in this chapter describe the methods implemented for the ray tracer used in this project. Ray tracing methods attempt to find irradiance falling onto each pixel in the *image plane* I. For every pixel at world position $\vec{o}_{ij}$ on I rays are cast from E, the *eye origin*. These rays are importance rays that will intersect and reflect or *hit* and *bounce* around the *scene*, described in section 3.1, according to the theory at section 3.2.

Taking into account the surface properties at each surface, consideration for the *radiance reflectance rate*, $f_r(\vec{x}, \hat{\omega}_i, \hat{\omega}_o)$, in section 3.3, is needed for evaluating the *radiance* at e.g. the point $\vec{p}$. The *light contributions* for said point $\vec{p}$ come in two forms, namely *direct light* (e.g. the light source L) and *indirect light* (e.g. the diffuse surface D). These are described in sections 3.4 and 3.5.



3.1 Scene Description

Below is a view of the room/scene that is to be rendered with ray tracing. The room has 6 walls, a floor and a roof. The floor and roof are Lambertian reflectors with the color white and all the walls are also Lambertian reflectors but with different colors, except for one which is a mirror (perfect reflector). The camera is placed at the origin facing the positive x-axis.

The light in the room comes from 2 additional triangles located on the roof facing the floor, forming a rectangle which acts as an area light.

Objects included in the scene are, a sphere that is a perfect reflector, and a polygonal tetrahedron with three mutually perpendicular faces that is a Lambertian reflector with the color cyan.

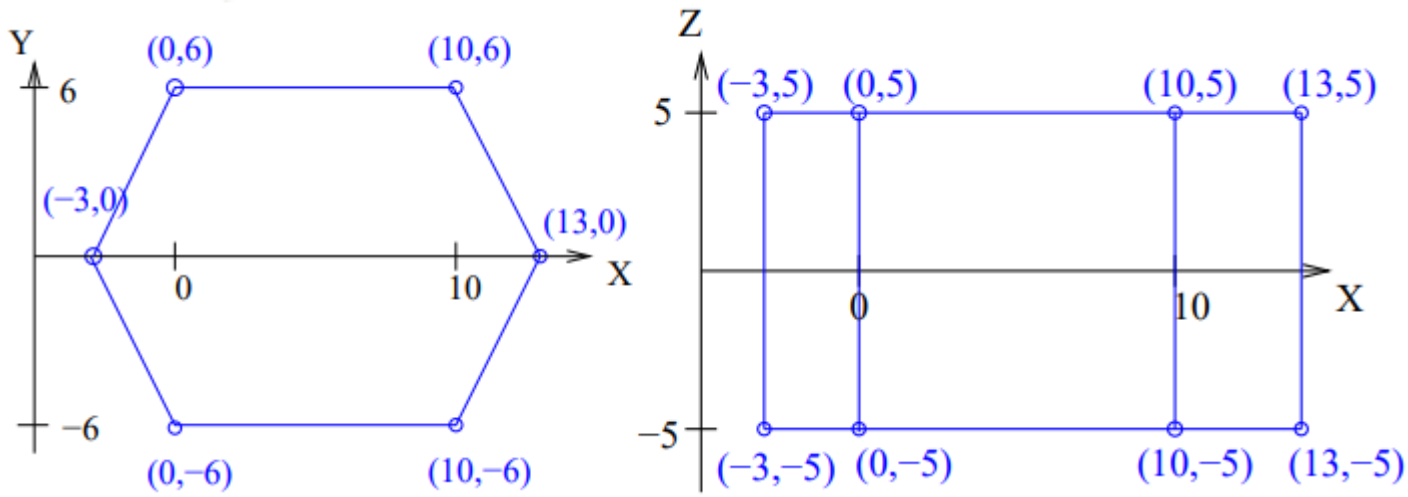


Figure 3.1: The world is a hexagonal room. The roof and floor are separated by the distance 10 along z and they lie in the planes z = 5 (roof) and z = 5 (floor). The walls are oriented orthogonally to the (x,y)-plane. The figure below shows the view of the flat from the top (left) and from the side (right).

3.2 Ray-Surface Intersections

In this project, the renderer must account for two types of ray-surface intersections; *Parametric Spheres* and *Polygonal Objects*. For the algorithms listed below, a ray is defined as a starting point ($\vec{\mathbf{r}}_o$) and a direction ($\vec{\mathbf{r}}_d$) i.e. a line of infinite length.

3.2.1 Parametric Spheres

Spherical objects are defined as a vector $\vec{\mathbf{s}}_{center} \in \mathbb{R}$ which denotes the origin of the sphere, and a scalar r as the radius of the sphere. Ray-Sphere intersections are done using a geometric solution to the following:

$$\mathbf{a} = (\hat{\mathbf{r}_d} \cdot \hat{\mathbf{r}_d}) \tag{3.1}$$

$$\mathbf{b} = 2\hat{\mathbf{r}_d} \cdot (\vec{\mathbf{r}_o} - \vec{\mathbf{s}}_{center}) \tag{3.2}$$

$$\mathbf{c} = (\vec{\mathbf{r}_o} - \vec{\mathbf{s}}_{center}) \cdot (\vec{\mathbf{r}_o} - \vec{\mathbf{s}}_{center}) - r^2 \tag{3.3}$$

$$\mathbf{d_{1,2}} = -\frac{\mathbf{b}}{2} \pm \sqrt{\left(\frac{\mathbf{b}}{2}\right) - ac} \tag{3.4}$$

If the value under the square root of equation 3.4 is less than zero, then there is no intersection between ray and sphere. Additionally in equation 3.4, if both terms $\mathbf{d_1}$ and $\mathbf{d_2}$ are equal to or less than zero, then there is also no intersection [5].

Otherwise, an intersection does exist and it would correspond to the following:

$$\vec{i} = \vec{\mathbf{r}_o} + min(\mathbf{d_1}, \mathbf{d_2}) * \hat{\mathbf{r}_d} \tag{3.5}$$

Moreover, the surface normal would then be found as $\hat{n} = ||\vec{i} - \vec{\mathbf{s}}_{center})||$.

3.2.2 Polygonal Objects

Intersections between triangles and rays are calculated using the *Möller-Trumbore* algorithm [5]. An algorithm which allows for intersection tests without the need for computing the plane containing the triangle. Using the corner vertices spanning up the triangle in question ($\vec{v_1}$, $\vec{v_2}$, $\vec{v_3}$) a local coordination system is computed in the form of:

$$\begin{aligned} \vec{e_1} &= \vec{v_2} - \vec{v_1} \\ \vec{e_2} &= \vec{v_3} - \vec{v_1} \end{aligned} \tag{3.6}$$

With $\hat{p} = \hat{\mathbf{r_d}} \times \vec{e_2}$, a check is made for if the ray travels parallel with the triangle. If the resulting determinant, found with $det = \vec{e_1} \cdot \hat{p}$, is equal to zero than the ray does not intersect with the triangle. Otherwise, further testing is done in the form of:

$$\begin{aligned} \vec{T} &= \vec{\mathbf{r}_o} - \vec{v_1} \\ \vec{Q} &= \vec{T} \times \vec{e_1} \\ u &= \vec{T} \cdot \hat{p} * det^{-1} \\ v &= \hat{\mathbf{r}_d} \cdot \vec{Q} * det^{-1} \end{aligned} \tag{3.7}$$

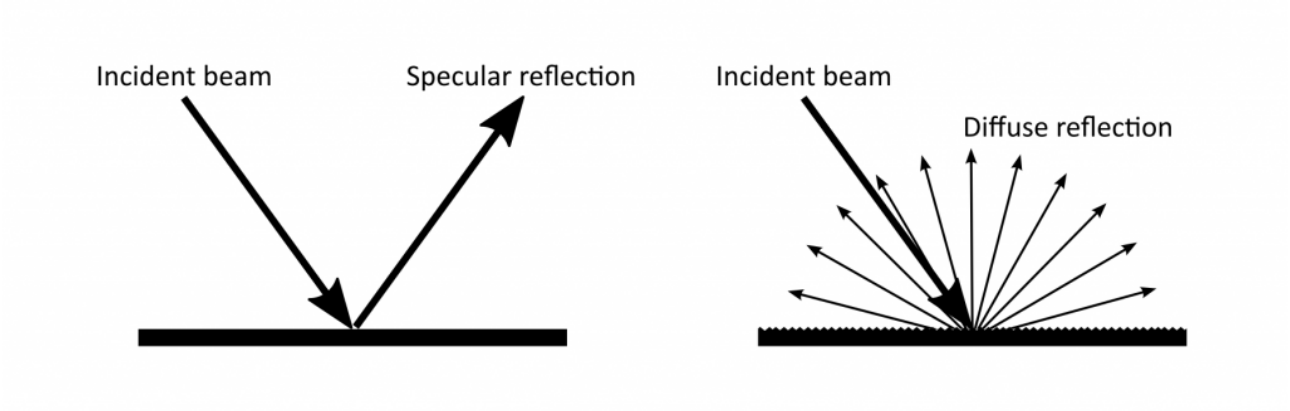
If $u \in (0, 1)$ and $v \geq 0$ and $u + v \leq 1$ are true, then an intersection is confirmed to have happened. The intersection point ($\vec{i}$) can then be found with:

$$\vec{i} = \vec{\mathbf{r}_o} + t * \hat{\mathbf{r}_d} \tag{3.8}$$

Where $t = \vec{e_2} \cdot \vec{Q} * det^{-1}$, is the travel distance from the ray origin to the intersection point.

3.3 Surface Properties

The definition of material properties for surfaces in the scene is what will determine an objects final appearance. For this render, only two types of materials used, namely “Diffuse” and “Specular”. The types react differently to light and emit it differently. For an overview on how they reflect light, see figure 3.2.



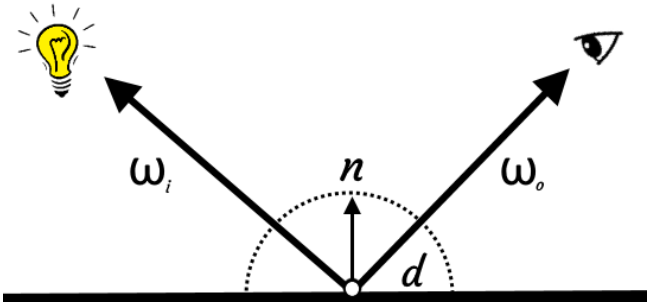


Figure 3.2: Diffuse Lambertian surface.

3.3.1 The Lambertian Model

The model for diffuse Lambertian surfaces reflects the light equally in all directions on the hemisphere. Meaning that, the reflected radiance is independent of the incoming/outgoing directions. Diffuse materials use the Lambertian BRDF seen in equation 3.9, which is constant in all directions within the hemisphere.

$$f_r(\vec{x}, \hat{\omega}_i, \hat{\omega}_o) = \frac{\rho_d}{\pi}, \tag{3.9}$$

the term ρ_d is the *albedo* of the Lambertian surface d, while π is the normalization factor for energy conservation. The albedo can simplistically be defined as the color of the surface.

3.4 Direct Light Contributions

As mentioned in section 2.4, most rays cast from the camera into the scene will never reach a light source. For this reason shadow rays are used, with the purpose of having direct light from light sources contribute to the total radiance at the intersection point.

Shadow rays work by sending a ray from each intersection point directly towards a random point on the light sources. Then, to determine if the intersection point lies in shadow, a visibility test is performed. The visibility test calculations can be computationally heavy but is important for a realistic rendering. Especially when trying to achieve realistic soft shadows from a more diffuse light source (area light source).

To get soft shadows, a predetermined amount of shadow rays are sent from each intersection, some may be occluded by objects whilst some manage to reach a point on the area light source. This means that some direct illumination contributes to the intersection point. The direct illumination calculation must be scaled by the number of shadow rays sent and the area of the light source, see equation 3.10.

$$directlight = directlight * \frac{\textbf{Area of light source}}{\textbf{Number of shadow rays}} \tag{3.10}$$

3.4.1 The Area Light Sources

Each intersection point will need to be approximate the direct light from an area light source. A Monte Carlo estimator $\langle L_D \rangle$ is used for this, with N shadow rays, see equation 3.11. Furthermore,

as mentioned in 3.1, the light source consists of triangles with corners v_1, v_2 and v_3 with the area $A = ||(v_3 - v_1) \times (v_2 - v_1)|| * 0.5$.

Before sending shadow rays, a destination must be determined. A uniformly random point q on the light source is needed to be selected through the probability distribution function $p(q) = \frac{1}{A}$.

This is achieved by drawing two random numbers $a, b \in (0, 1)$ until $a + b > 1$ and creating the point q with barycentric coordinates $q = (1 - a - b)v_1 + av_2 + bv_3$. This process is to be repeated until N points are created. Since the light source is a Lambertian emitter, L_0 is emitted for all points and directions of the light source.

$$\langle L_D \rangle = \frac{AL_0}{N} \sum_{i=1}^{N} f_r V(x, q) G(x, q_i) \tag{3.11}$$

The term $V(x, q)$ is defined as the visibility function which results in the value 1 if no object occludes the path to the light point and 0 otherwise.

The term $G(x, q_i)$ is described as the geometric term and can be found with $\frac{cos\alpha cos\beta}{d^2}$, where d is the distance to the light point; α and β are the inclination angles to the surface normal at the start- and endpoint.

3.5 Indirect Light Contributions

When rays are emitted from the camera they will intersect with surfaces. At the intersections the rays will either terminate or spawn new rays depending on the surface material. By following the path of the ray a tree of sorts is built up with intersections as nodes connected by rays. The child nodes will contribute radiance to the parent nodes which gives indirect light, i.e. light reflected at least once since it left the light source. Light reflections are usually split it into three types, however, this render only has to deal with two; specular and diffuse reflections.

3.5.1 Specular Reflections

Once a ray intersects with a perfect specular surface (mirror), a new ray is always spawned (won't terminate on a mirror surface). The direction of the new ray is $\vec{R} = \vec{I} - 2(\vec{I} \cdot \vec{N})\vec{N}$, where $\vec{I}$ is the incoming ray and $\vec{N}$ is the surface normal.

3.5.2 Diffuse Reflections

As mentioned in 3.3.1, the project only uses the Lambertian model for diffuse surfaces. Once a ray intersects with a diffuse surface a new direction is to be determined. This is done be choosing a random azimuth angle (ϕ_i) and inclination angle (θ_i) in the hemisphere. Two random values $u, v \in [0, 1)$ are generated and with equations 3.12 and 3.13 new azimuth and inclination angles are computed. The reflected ray will contribute to the radiance at the intersection transmitting indirect light.

$$\phi_i = 2\pi u \tag{3.12}$$

$$\theta_i = cos^{-1}(\sqrt{v}) \tag{3.13}$$

3.5.3 Russian Roulette

In order to avoid a biased result, Russian Roulette is used for getting an unbiased termination condition. Meaning that it keeps the renderer unbiased while still keeping ray paths from becoming too long. It is in practice a very simple solution to ray termination. For our specific implementation we have it so that every recursive iteration of a ray will either be terminated or be reflected if a uniform random number $r \in (0, 1)$ is lower than $\alpha \in (0, 1)$, a value which is defined as the maximum radiance or importance at the current point, more specifically the larges value amongst the points color vector. Making the reflection probability connected to the surface properties.

Chapter 4

Results and Benchmarks

This chapter present the results from the implementation and benchmark results from rendering with different settings. As mentioned in 3.1, the scene is made up by a room with six corners with a roof and floor. The walls all have different colours where one wall is specular to act as a mirror. Both the roof and floor is set to be white. In the room there are two objects, one reflective sphere with specular reflection and one tetrahedron with lambertian reflection. The last thing i the scene is a rectangular light in the roof which is the only and primary light source in the scene.

4.1 Image results

Four final images were rendered all with a resolution of 800 by 800 pixels. The settings that were altered between renders were *spp* (sample per pixel) and the number of shadow rays per ray bounce (*NoSR*). All renders where done with a max depth of 5 labertian bounces. The images can be seen in the figures 4.1, 4.2, 4.3 and 4.4.

4.2 Render benchmarks

The rendering of the images were done desktop PC with a Intel i7-4770K CPU with a clock speed of 3.50GHz. No multi-threading was implemented for the program, the benchmark times are for sequential thread execution.

Table 4.1: Table displaying render benchmarks.

Figure	spp	NoSR	Time [min]
4.1	50	2	218
4.2	25	2	114
4.3	5	1	16
4.4	5	10	87

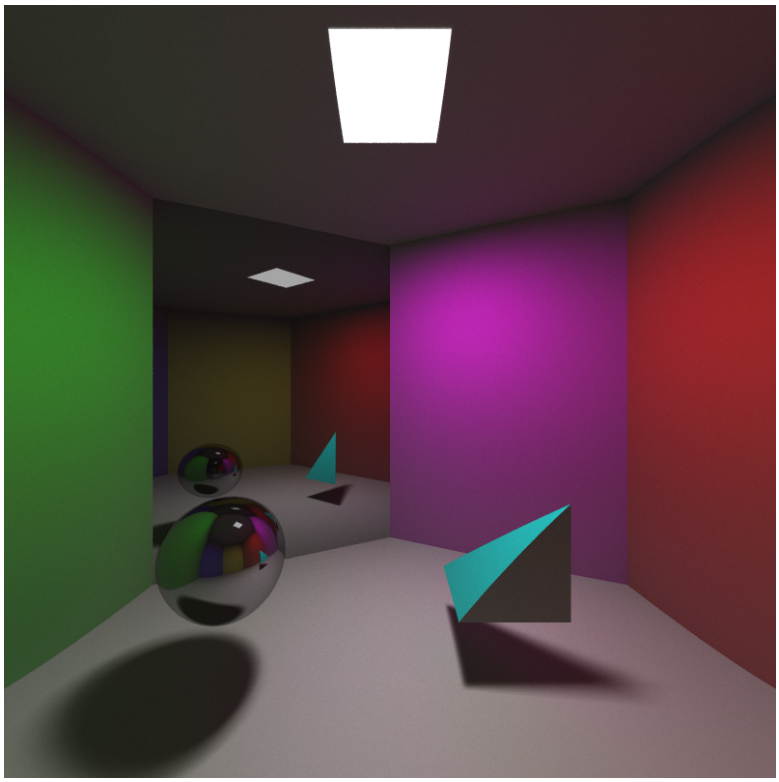


Figure 4.1: Ray traced 800x800 image with 50 spp and 2 shadow rays.

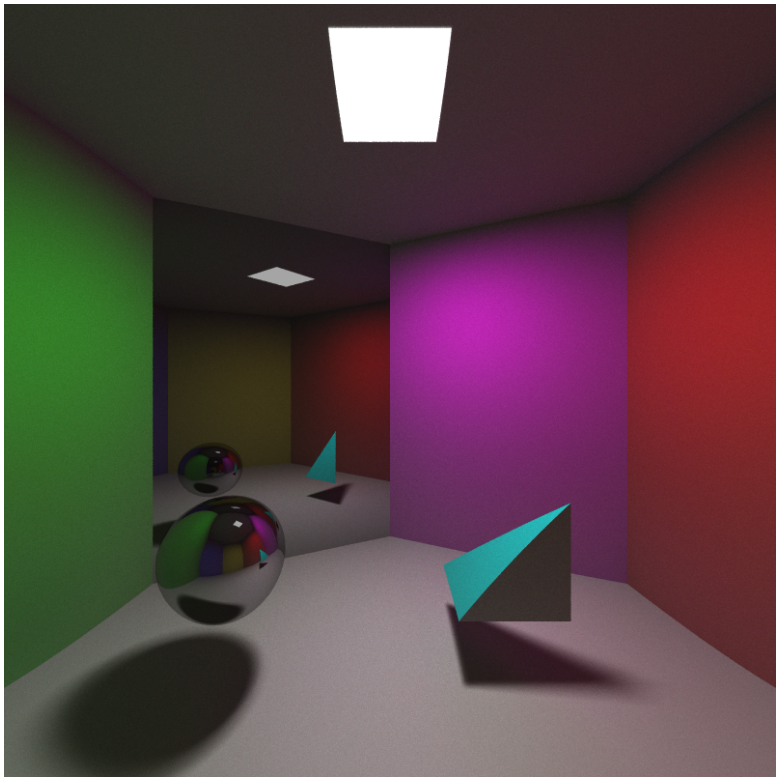


Figure 4.2: Ray traced 800x800 image with 25 spp and 2 shadow rays.

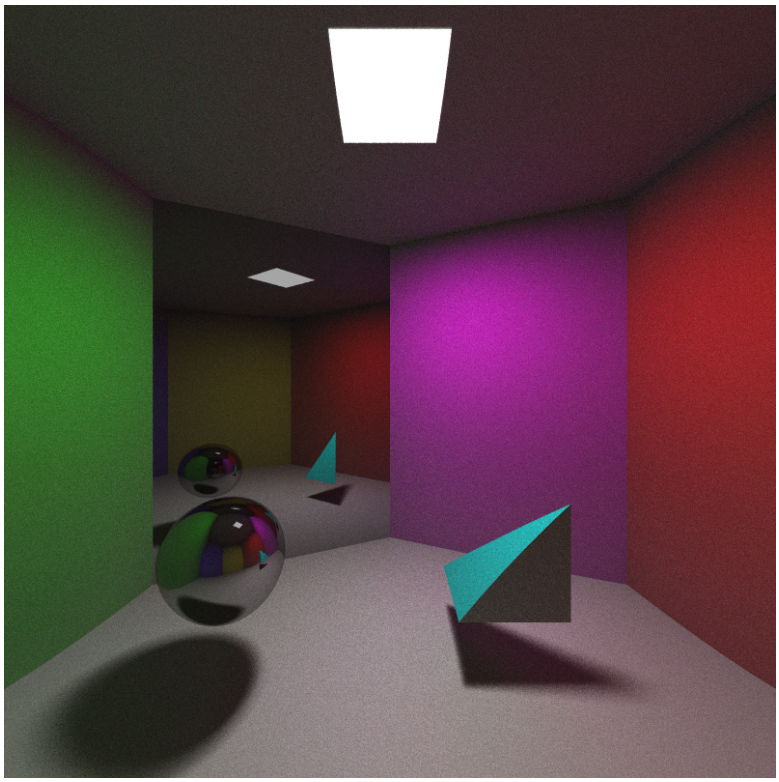


Figure 4.3: Ray traced 800x800 image with 5 spp and 1 shadow rays.

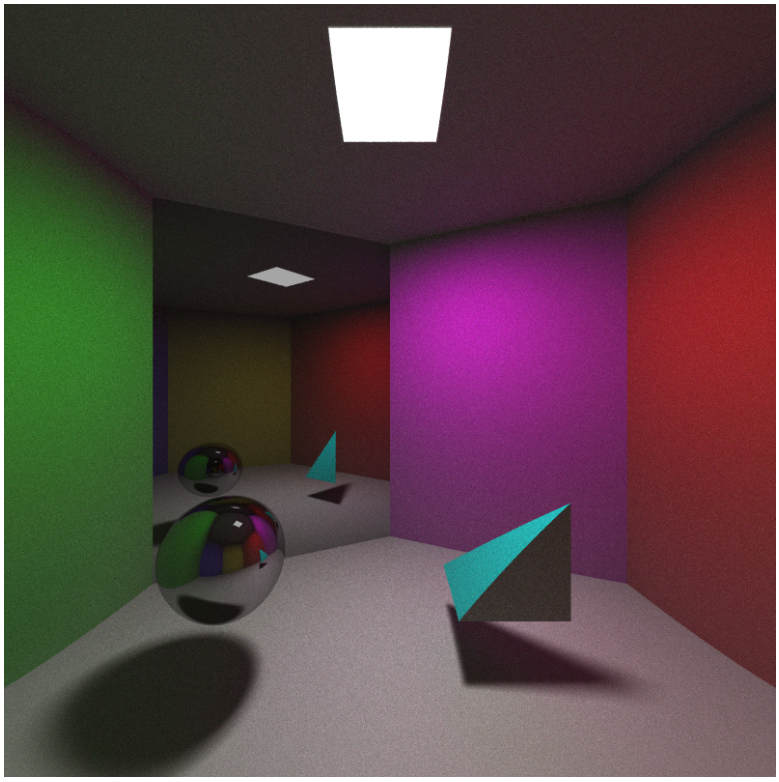


Figure 4.4: Ray traced 800x800 image with 5 spp and 10 shadow rays.

Chapter 5

Discussion and conclusion

This chapter includes a broad discussions about the different ray tracing method, the implementation and results. Where known errors and faults in the implementation is highlighted. In the end there will be a short conclusion of the results.

5.1 Different ray tracing methods

As described in previous chapters are there advantages and disadvantages with all ray tracing methods. The main once are time and photo realism. A fast method has a hard time to be photo real and a photo real method has a hard time to be fast. When it comes to the Monte Carlo method because it uses random samples it need several samples per pixels in order to have an acceptable noise level. Despite being slow it is not perfect either. Monte Carlo can not really simulate caustics because the random samples makes it very unlikely that a ray bounces up thru a object and then hit the light. Other methods like photon mapping handles caustics better than Monte Carlo due to the fact that the rays are shoot from the light not the camera.

5.2 Implementation

The implementations for this project is just one of many ways of implementing a Monte Carlo ray tracer, which makes it hard to evaluate the features and quality of the image by just looking at the code. One small difference can have a massive impact on the rendered image. Otherwise does the implementation follow known methods and mathematical formulas regarding Monte Carlo ray tracing.

5.3 Results and known issues

The final render images shows the characteristics of a ray traced image with reflections, colour bleeding and not completely dark roof or shadows due to bouncing rays that carry light information. From the table 4.1 is it easy to see the impact samples per pixel has on the rendering time. The image with 50 spp look really nice with a low noise level but the image with 25 spp looks very similar and it took about half the time to render. With the 5 spp images is the noise very visible and not suitable for any commercial work. Using different numbers of shadow rays did no affect the shadows in any visible way it only took longer to render. This lack of difference can be a result of the low spp value of 5 but most likely is it due to the simple scene with just one light that does no crate complicated shadows to

begin with.

This implementation is by no means a perfect Monte Carlo ray tracer and do have some known issues. The mirror wall is darker than the actual scene, this can have something to do with a set reflection coefficient that acts due to recursion. Further investigation and debugging on this issue was not prioritised and is still an issue. Another issue that impacts the image visuals is related to the shadows. When a point in the scene is deemed to be in shadow thru the shadow ray test is the point in question set to black. Later when bouncing light is added making the shadow less black is the original colour of the surface that is in shadow lost. The fault can be a result of a not perfect integration of the shadow ray test in the cast ray recursive function in the implementation. This issue was noticed during final render tests at higher resolution and spp. No fast fix was found so the issue is still a part of the implementation.

5.4 Conclusion

In the end has a basic Monte Carlo ray tracer been implemented that produces images with global illumination. There are things that could be improved for the visuals like the shadow colour problem and the mirror. The rendering time can be improved by implementing multi-threading in the program in order to execute several ray tracings in parallel. But aside these issues does the implantation work well and consistent.

Bibliography

[1] Goral Cindy, Torrance Kenneth, Greenberg Donald, Battaile Bennett.”Modelling the interaction of light between diffuse surfaces”. SIGGRAPH ’84 Proceedings, Cornell University (1984).

[2] Appel, Arthur. "Some techniques for shading machine renderings of solids". IBM Research Center (1968). Hämtad från:
http://graphics.stanford.edu/courses/Appel.pdf

[3] Cook, R. Porter, T. Carpenter, L. ”Distributed Ray Tracing.”. Computer Graphics 18:3 (1984).

[4] Dieckmann, Mark E. "TNCG15: Advanced Global Illumination and Rendering, Lecture 7 (Probability theory)" (2018). Hämtad från:
https://liuonline.sharepoint.com/sites/TNCG15/TNCG15_2018HT_1T/CourseDocuments/Forms/AllItems.aspx

[5] Dieckmann, Mark E. "TNCG15: Advanced Global Illumination and Rendering, Lecture 6 (The scene)" (2018). Hämtad från:
https://liuonline.sharepoint.com/sites/TNCG15/TNCG15_2018HT_1T/CourseDocuments/Forms/AllItems.aspx

[6] Dieckmann, Mark E. "TNCG15: Advanced Global Illumination and Rendering, Lecture 8 (Monte-Carlo integration)" (2018). Hämtad från:
https://liuonline.sharepoint.com/sites/TNCG15/TNCG15_2018HT_1T/CourseDocuments/Forms/AllItems.aspx

[7] Dieckmann, Mark E. "TNCG15: Advanced Global Illumination and Rendering, Lecture 11 (Rendering equation)" (2018). Hämtad från:
https://liuonline.sharepoint.com/sites/TNCG15/TNCG15_2018HT_1T/CourseDocuments/Forms/AllItems.aspx

[8] Dieckmann, Mark E. "TNCG15: Advanced Global Illumination andRendering, Lecture 3 (Radiosity)" (2018). Hämtad från:
https://liuonline.sharepoint.com/sites/TNCG15/TNCG15_2018HT_1T/CourseDocuments/Forms/AllItems.aspx

[9] Wann Jensen, Henrik. “Global Illumination using Photon Maps”. Department of Graphical Communication, The Technical University of Denmark (1996).

[10] Whitted, Turner. “An improved illumination model for shaded display.” SIGGRAPH (1979).