Monte Carlo Ray Tracing

TNCG15 Advanced Global Illumination and Rendering

Måns Aronsson, Filip Arvidsson

October 29, 2021

Contents

Abstract

A Monte Carlo ray tracer is implemented to demonstrate theoretical and practical knowledge about global illumination and rendering. The selected approach is a combination of Whitted ray tracing and radiosity. The implemented renderer can render triangles, boxes and spheres with diffuse, mirror and glass materials. The influence of indirect illumination are visible as color bleeding, caustics, and the illumination of areas occluded from light sources. Image quality is substantially improved by increasing the number of samples per pixel.

Chapter 1

Introduction

The goal of most rendering applications is to create images of virtual worlds, as realistically as possible, within a certain time frame. Many shortcuts have to be taken if the goal is to create many such images per second. If instead a much longer time frame is permitted, many options on how to produce these images become available. This usually entails some method of *global illumination*. Global illumination models strive to mimic the physical properties of real light, meaning, it has been emitted, reflected, refracted, absorbed and re-emitted an incomprehensible amount of times before it finally finds its way into your eye and triggers a neural reaction. A renderer needs to simplify this substantially, but even with extensive simplifications of real world physics, global illumination methods have proven to be capable of producing images indistinguishable from the real world.

The aim of this project is to learn the global illumination techniques of *Monte Carlo ray tracing* and demonstrate it by implementing a renderer. The report covers both some of the basic concepts of global illumination techniques, and how these were implemented in code.

Chapter 2

Background

This chapter covers some basic concepts of global illumination and physics that are needed to understand what a Monte Carlo ray tracer is and how it can be implemented.

2.1 Whitted ray tracing

Whitted ray tracing is the method of creating photorealistic images by sending out rays through the camera as shown in Figure 2.1, instead of sending light from light sources and hoping the photons will hit the observer. The rays have their origin in the observer and a direction in which they travel through the scene. The direction of each ray depends on a pixel, which is what the image will consist of. The color of the pixel will be determined by the object which the ray hits first. There are a number of methods of determining the color, but Whitted ray tracing makes the process fast and efficient, in contrast to letting light travel as it would in the real world.

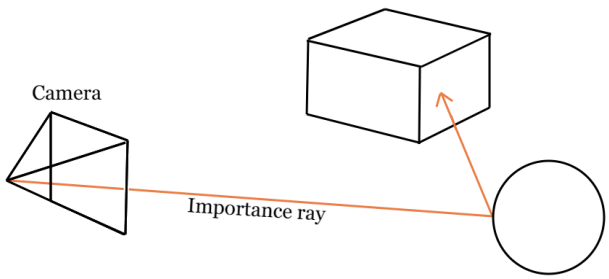


Figure 2.1: Reflected direction depending on incoming direction and the object normal.

2.2 Radiosity

Radiosity is a technique for lighting a scene with indirect illumination. Areas which are not under direct illumination when observed in reality are usually still visible. Their visibility is made possible by light traveling from the light source and bouncing on other surfaces. The indirect illumination calculated through radiosity enables more realistic imagery by lighting up a scene without unrealistic ambient illumination.

Radiosity is achieved by dividing surfaces into smaller patches and determining how much light is transferred off them. The light leaving a patch is a combination of the light being emitted and reflected by

the surface. The radiosity method is independent from the viewpoint of the observer.

2.3 Monte Carlo estimation

Many complex problems have impossible or laborious analytical solutions. Instead, it might be a much simpler task to define the task itself than its solution. The Monte Carlo method estimates the correct solution using the means of random sampling of the task's action space. The more samples that are used, the lower variance the solution will have.

In the case of global illumination, the light of each point is a combination of the direct illumination of the light sources in the scene and light that have bounced from other surfaces before hitting that point. The global light contribution can be seen as an integral of all incoming rays in a hemisphere around the point. In a Monte Carlo ray tracer, that integral is estimated using a Monte Carlo method.

2.4 Bi-directional Reflectance Distribution Functions (BRDFs)

In the physical world, energy is never lost but just transformed into new forms. Photons striking an object surface might bounce on that surface while photons of other wavelengths get absorbed and are transformed to heat. If there are atoms nearby, photons are constantly changing direction, getting absorbed and re-emitted. The complexity of material properties in the physical world must be simplified in a renderer by using BRDFs. More on this in 3.5.

2.4.1 Ideal reflection

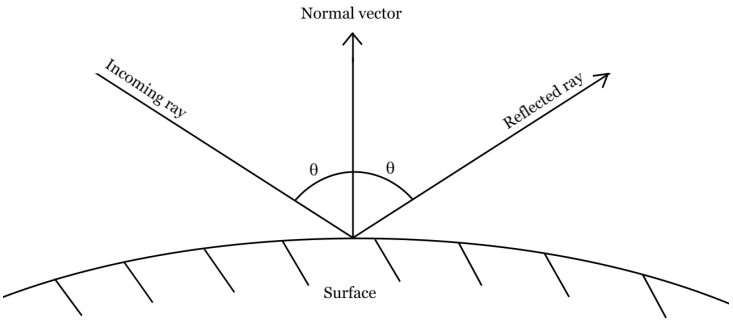


Figure 2.2: Reflected direction depending on incoming direction and the object normal.

An ideal reflection refers to when energy is preserved and the angle of the outgoing ray is the same as the angle of the incoming angle in relation to the surface normal, see Figure 2.2. This results in a mirror-like effect if the entire surface perfectly reflects incoming rays. In the physical world, even mirrors absorb some of the incoming light energy. However, this is simplified in this renderer's BRDF.

2.4.2 Diffuse reflection

Zooming in on a highly frictional surface, it would become apparent that the friction originates from its rugged property. This has a great impact on reflections as well. The surface normal where the light hits might be in a completely different direction than its apparent curvature. These kinds of surfaces are

called diffuse surfaces, see Figure 2.3. A diffuse reflection scatters its outgoing rays in all directions and absorbs some wavelengths of the light, resulting in the experience of color that can *bleed* to other close objects.

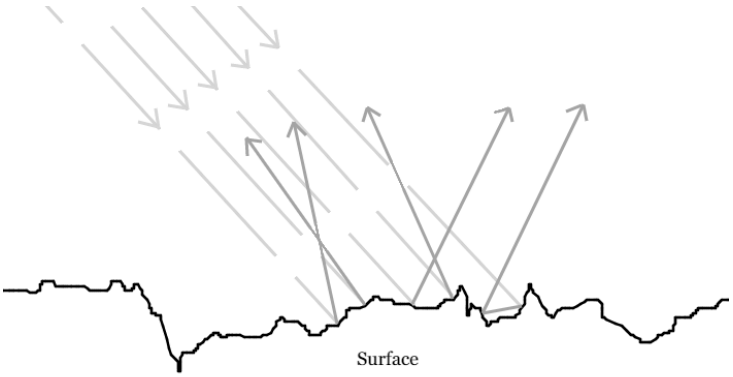


Figure 2.3: Incoming rays being reflected in random directions on a diffuse surface.

2.4.3 Refraction

Transparent materials let light go through. However, the light normally wont go straight through but instead change direction as it enters and leaves the transparent medium, i.e. the light is being refracted. The new light angle follows Snell's law, meaning it depends on the differences in refractive index between the medium before the intersection and after the intersection, e.g. air and glass [1]. Depending on the incoming lights angle towards the object normal, parts or all of the light will be reflected instead or refracted, see Figure 2.4.

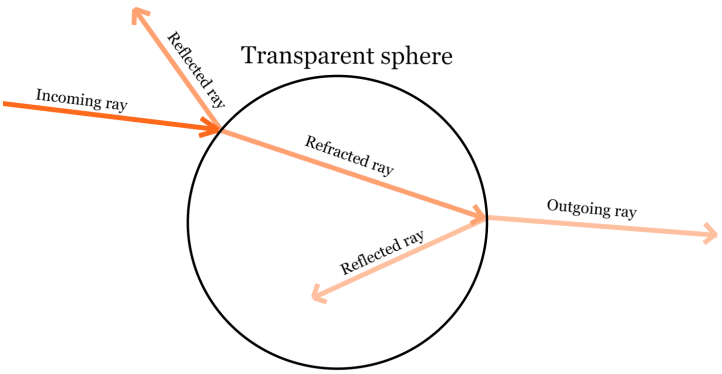


Figure 2.4: Incoming ray being reflected and refracted intersecting with a transparent object.

2.5 The rendering equation

A point in 3D space is illuminated directly by the light sources in the scene, by reflected and refracted light. This is done by the rendering equation 2.1,

$$L(x \to \omega_{out}) = L_e(x \to \omega_{out}) + \int_{\Omega} f_r(x, \omega_{in}, \omega_{out}) L_D(x \leftarrow \omega_{in}) \cos\theta_{in} \, d\omega_{in} \tag{2.1}$$

where L is the resulting radiance of point x, L_e is self emitted radiance if point x is on a light source, the integral $\int_{\Omega}$ represents the sampled rays in a hemisphere around point x as discussed in 2.3, f_r is the material properties obtained from the BRDFs, L_D is the incoming radiance from bouncing rays

and $cos\,\theta_{in}$ is the geometric term that makes incoming radiance contribution angle dependant [5]. The minimum value of the cos-term is set to zero, as incoming radiance can not be negative. The final result is obtained by solving the rendering equation for all pixels in the image.

2.6 Ray intersection

Rays sent out into the scene will intersect with at least one object, since the scene is a closed room. The method for finding potential intersection points differs depending on the mesh. Intersection points are found by describing positions of surfaces as an equation, and interpolating between the start- and end-point of a ray. The interpolation between a ray's start and end is defined in Equation 2.2, where $x(t)$ can describe any point along the ray.

$$x(t) = p_s - t(p_e - p_s) \tag{2.2}$$

where t is the interpolation value which the ray is multiplied with to get the intersection point, p_s is the origin and p_e the end of a ray.

2.6.1 Triangle intersection

The intersection between a triangle surface and a ray is calculated using the Möller-Trumbore algorithm [2]. Any point on the surface of a triangle can be given by Equation 2.3.

$$T(u, v) = (1 - u - v)v_0 + uv_1 + vv_2 \tag{2.3}$$

where T is a point on the triangle and v_0, v_1 and v_3 are vertices. Combining Equation 2.2 with Equation 2.3 and solving it using Cramer's rule gives the solution for t in Equation 2.4 [4].

$$t = (Q \cdot E_2)/(P \cdot E_1) \tag{2.4}$$

$$\begin{aligned} E_1 &= v_1 - v_0 \\ E_2 &= v_2 - v_0 \\ Q &= (p_s - v_0) \times E_2 \\ P &= (p_e - p_s) \times E_2 \end{aligned}$$

where E_1 and E_2 are edges of the triangle, Q is the vector orthogonal to E_2 and the vector between the first vertex and the ray's starting point, and P is the orthogonal vector between E_2 and the incoming ray. Cramer's rule will also give the value of the u,v-coordinates of the triangle. The u, v and t might not be valid. Intersection occurs if the following criterias are met:

$$\begin{aligned} u &\geq 0 \\ v &\geq 0 \\ u + v &\leq 1 \\ t &\geq 0 \end{aligned}$$

If t was allowed to be negative, the camera would display objects it intersects behind it.

2.6.2 Sphere intersection

Any point on the surface of a sphere can be described by Eqaution 2.5.

$$(o + tl - c)^2 = r^2 \tag{2.5}$$

where o is the starting point of the ray, t corresponds to t in Equation 2.4, l is the normalized direction of the ray, c is the center point of the sphere and r the radius. This can be rewritten as a second degree polynomial and solved using the pq-formula, as shown is Equation 2.6.

$$t = -\frac{2l \cdot (o - c)}{2} \pm \sqrt{\left(\frac{2l \cdot (o - c)}{2}\right)^2 - ac} \tag{2.6}$$

The method will return two intersection points as the sphere is a solid object. The first intersection is the one desired, which will be the smallest t returned, if both t are larger than zero.

2.7 Shadow rays

Shadow rays are used to determine if a point is under direct illumination from a light source, as visualized in Figure 2.5. When there is a ray intersection with an object, a shadow ray is sent out in the direction of the light source. If the shadow ray intersects another object before the light source, the intersection point lays in shadow and receives no direct illumination.

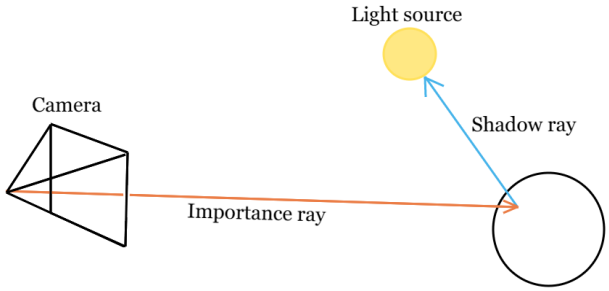


Figure 2.5: Shadow ray sent out from intersection point between ray and sphere.

The light source can be either a point- or area light. If it's a point-light, the shadow ray will have one possible endpoint. In the case of area lights, shadow rays can have multiple possible endpoints.

2.8 Light intensity

The intensity given off by a light source to a point is inverse-squared to the distance between them. As the distance increases, the amount of light received will subside rapidly according to The Inverse Square Law [3].

2.9 Anti aliasing

Anti aliasing is a method of smoothing out pictures by filling in missing information between data points. A problem with ray tracing is that the picture can appear sharp and with artifacts. These mishaps occur because of the limited resolution size of the view plane and that pixels close to each other can receive vastly different values. A method to counter the jaggedness of the picture is an anti aliasing technique called supersampling. The technique works by rendering a pixel value several times and then averaging the result. The pixel is split up into several smaller areas, where one ray is sent through a random point in each area, as shown in Figure 2.6. All these areas will contribute to the final value and can be looped through several times, creating more samples per pixel.

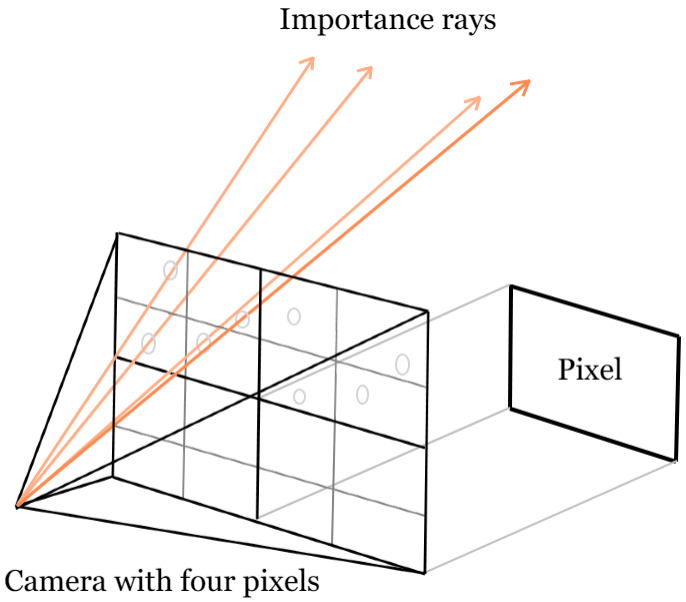


Figure 2.6: A camera with four pixels. Four rays are sent out with a randomized direction defined within each quarter of the pixel.

Chapter 3

Method

The following chapter explains how the topics covered in Chapter 2 were used and implemented in the project. All code was written in *C++17* and compiled with *Visual Studio 2019*.

3.1 The scene

The project featured a closed hexagonal room, consisting of walls made from triangles. Inside the room were two types of objects: polygons and spheres. Located at the top of the scene was an area light. The light was composed of two triangles forming a square, which brought illumination to the scene. The room also featured a camera, along with a view plane divided into pixels, in order to visualize the scene.

The scene held two containers of pointers to objects, one for all objects and one for area lights. Through the containers, the scene could assign targets to which a ray would first intersect and use the information to calculate the radiance of the ray, see 2.5.

3.1.1 Illuminating the scene

When a ray intersected an object, shadow rays were sent towards the area light with its origin placed outside the object. This offset of the origin was made by adding 4% of the object's normal where the ray intersected. The direction of the ray was made by randomizing an end point using the *u,v*-coordinates of the area light.

One shadow ray was sent to each light source. The direct illumination was calculated by averaging the result, adding all illumination together and dividing by the total amount of shadow rays and number of light sources in the scene, as shown in Equation 3.1. No shadow rays were sent out if a ray hit a light source.

$$\Theta_r = \sum_{i=1}^{N} \frac{(\alpha \cdot \beta \cdot \Theta_i)}{\|s_i\|^2 \cdot N} \tag{3.1}$$

$$\alpha = -\widehat{D}_s \cdot \widehat{N}_l$$
$$\beta = \widehat{D}_s \cdot \widehat{N}_o$$

where Θ_r is the amount of reflected light from the direct illumination, N is the number of light sources, Θ_i is the amount of light emitted from the light source, s_i is the shadow ray sent to the light source as

described in 2.8, α is the dot product between the the negative normalized direction of the shadow ray D_s and the normal of the light source, β is the dot product the normalized shadow ray direction and the normal of the intersected object. The product between α and β needs to be larger than one, otherwise it is set to zero. The reason is to not produce any negative light, as they are the only terms which can be negative. Negative light can be a consequence of the light illuminating from behind or an object facing away from the light source.

3.1.2 Indirect illumination

The radiance rays consisted of a local light contribution and a global one, except the leaf nodes of a ray tree that only had a local one. The ray tree was created by evaluating one branch at a time until a leaf node was hit. When all children of the current node were leaves, that node's radiance was evaluated, the child nodes were deleted and that node was then set to be a leaf node. This was then repeated for all branches of the tree until the root was evaluated. The equation for evaluation a node was

$$L_{tot} = L + \sum(L_c\, w_c)/w \tag{3.2}$$

where L_{tot} is the evaluated radiance, L is the local light contribution, L_c is the radiance of a child ray, w_c is the importance of the same child and w is the importance of current ray.

3.2 The camera

A camera class was implemented that contained a plane, consisting of the four vertices (0, 1, 1), (0, -1, 1), (0, 1, -1) and (0, -1, -1), two additional vertices defining observer positions, and an array of pixels. Rendering functionality was implemented where the color of each pixel in the array was set by sending many rays, see 3.4, through each pixel in the camera plane, see 2.9, and then trace it through the scene until its importance got below a threshold. The result for each pixel was then averaged out by the number of rays that have been sent through the pixel.

Functionality for image creation was implemented where the values of all the pixels firstly were balanced by taking its square root instead, making bright and dark pixels closer in value. The pixel values were then converted to RGB-values between 0-255 and outputted as a bitmap-image.

3.3 Object

A base class called *Object* was introduced. Through this class, a number of virtual functions could be accessed by different kinds of objects. The function for getting the normal vector, ray intersection and the method for generating shadow rays was shared and could be called by all objects, as they were derived from the base class.

3.3.1 Polygon

A triangle consisted of three vertices, ordered such that the normal faced outwards. The triangles were used to build walls, boxes and area lights. Boxes were created by generating twelve triangles, based on the desired height, width, length and a center point. The triangles which made up the cube was added to the array of objects, which enabled them to be rendered through triangle intersection as shown in Equation 2.4.

3.3.2 Sphere

The spheres were not built up by triangles like polygons, but instead instantiated using a center point with a defined sphere radius.

3.4 Ray

Rays consisted of four vertices, *start*, *end*, *direction* and *radiance*, an importance value as well as pointers to the object that intersected it, its parent and its children rays. It also stored information about what depth it was in the ray tree, if it was a leaf node in the ray tree and if it was inside of a transparent object. The start, direction and importance of the root in each ray tree was set in the camera 3.2, and after that set in the BRDFs of target object materials 3.5 together with parent and children pointers. The ray tree depth was needed to ensure that no infinite reflection scenario between two or more mirrors, or inside transparent objects, would occur. The leaf information was needed for the indirect illumination algorithm to work, see 3.1.2. The information about being inside a transparent object was needed for correct calculations of the glass BRDF, see 3.5.4.

3.5 Material

The material class was implemented with four sub-classes. The base class held the material color and emittance and defined that all sub-classes needed BRDF-functionality, see 2.4. The BRDFs returned one or many reflected and refracted rays where the total outgoing importance of the rays was equal or less to the incoming importance. The implemented sub-classes were *Mirror*, *Diffuse Lambertian*, *Light* and *Glass*.

3.5.1 Mirror

The mirror BRDF returned one ideally reflected ray, see 2.4.1. The outgoing importance of the reflected ray was the same as for the incoming ray. A ray would never stop on a mirror surface except if it was stuck in an infinite reflective loop and thus exceeded the predefined maximum ray depth.

3.5.2 Diffuse Lambertian

The diffuse Lambertian BRDF returned one or many reflected rays. As discussed in 2.4.2, each reflected ray had a random direction in a hemisphere around the normal of the diffuse surface. The total importance of the reflected rays were lower or equal to the importance of the incoming ray and depended on the reflectivity of the diffuse material. With random outgoing ray directions, the BRDF was independent of both view and light direction which resulted in a constant value of its integral [5]. Thus, the rendering equation 2.1 for diffuse surfaces became

$$L(x \to \omega_{out}) = L_e(x \to \omega_{out}) + \pi \int_{\Omega} \frac{\rho}{\pi} L_D(x \leftarrow \omega_{in}) \cos\theta_{in} \, d\omega_{in} \tag{3.3}$$

where L is the resulting radiance of point x, L_e is self emitted radiance if point x is on a light source, the integral $\int_\omega$ represents the sampled rays in a hemisphere around point x as discussed in 2.3, ρ [0, 1] is the reflectivity of the diffuse material, L_D is the incoming radiance from bouncing rays and $cos\,\theta_{in}$ is the geometric term that makes incoming radiance contribution angle dependant.

Rays had a chance to stop with every diffuse bounce by a technique called Russian Roulette. The

materials absorption was defined as

$$\alpha = \alpha_{min} - (\alpha_{max} - \alpha_{min}) \frac{||color_{object}||}{||color_{max}||} \tag{3.4}$$

where α is the absorption, α_{min} [1, ∞] and α_{max} [α_{min}, ∞] are predefined absorption boundaries, $||color_{object}||$ is the length of the object's color vector and $||color_{max}||$ is the maximum length of a color vector (white). Each diffuse bounce then had a probability to stop equal to $1 - \frac{1}{\alpha}$

3.5.3 Light

The light BRDF returned one ray with its importance set to zero. This made rays always stop when they reached a light source. The rendering of the light source was instead dealt with in the local lighting functionality in the scene object, see 3.1.

3.5.4 Glass

The glass BRDF returned two rays, one reflected and one refracted. The reflected ray's direction is that of an ideal reflection, see 2.4.1, and the refracted ray follows Snell's law, see 2.4.3. Any ray inside of a glass object only had a reflected part if the incoming angle is larger than the angle given by

$$\theta_{max} = \arcsin(\frac{n_1}{n_2}) \ , \ n_1 < n_2 \tag{3.5}$$

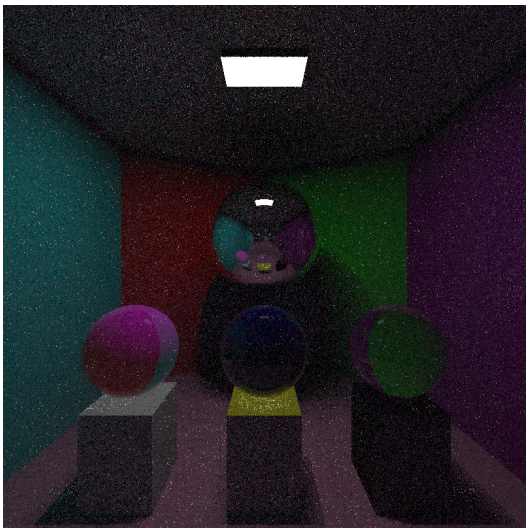
where θ_{max} is the angle, n_1 is the refractive index of the outside media (air), and n_2 is the refractive index of glass. Otherwise, the importance of the incoming ray was distributed to the reflected and refracted ray by *Schlick's approximation* formula,

$$\begin{aligned} R(\theta) &= R_0 + (1 - R_0)(1 - cos\,\theta)^5 \\ R_0 &= (\tfrac{n_1 - n_2}{n_1 + n_2})^2 \end{aligned} \tag{3.6}$$

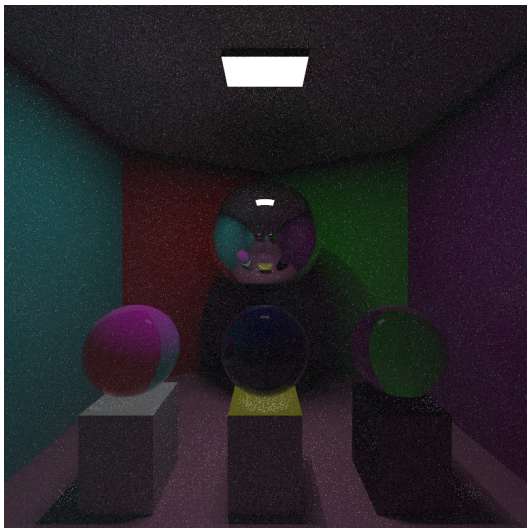
where $R(\theta)$ is the reflected ratio, θ is the incoming ray's angle to the surface normal, n_1 is the refractive index of the media where ray is coming from and n_2 is the refractive index of the media the refracted ray enters. Then, the refracted ratio of the incoming importance becomes $1 - R(\theta)$.

Chapter 4

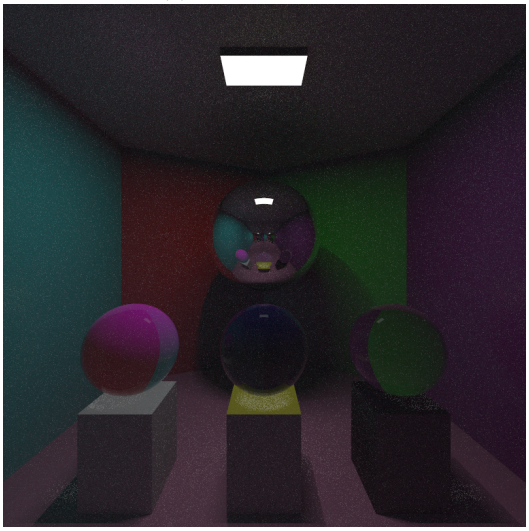
Results



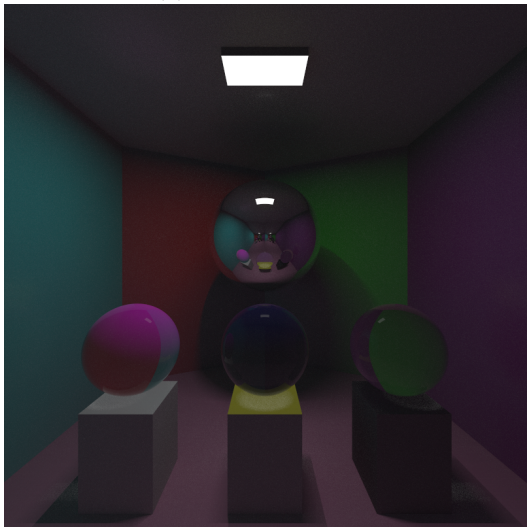
(a) 1 sample per pixel



(b) 4 samples per pixel



(c) 16 samples per pixel



(d) 256 samples per pixel

Figure 4.1: Four renders with 800x800 pixels of the same scene with different number of ray samples per pixel. The scene consists of a white area light, a mirror sphere, mirror walls behind the camera with the rest of the room being diffuse, three diffuse boxes, a colorless glass sphere $(0.0, 0.0, 0.0)$ to the right, a light blue glass sphere $(0.0, 0.0, 0.1)$ in the middle, and a purple glass sphere $(1.0, 0.0, 1.0)$ to the left.

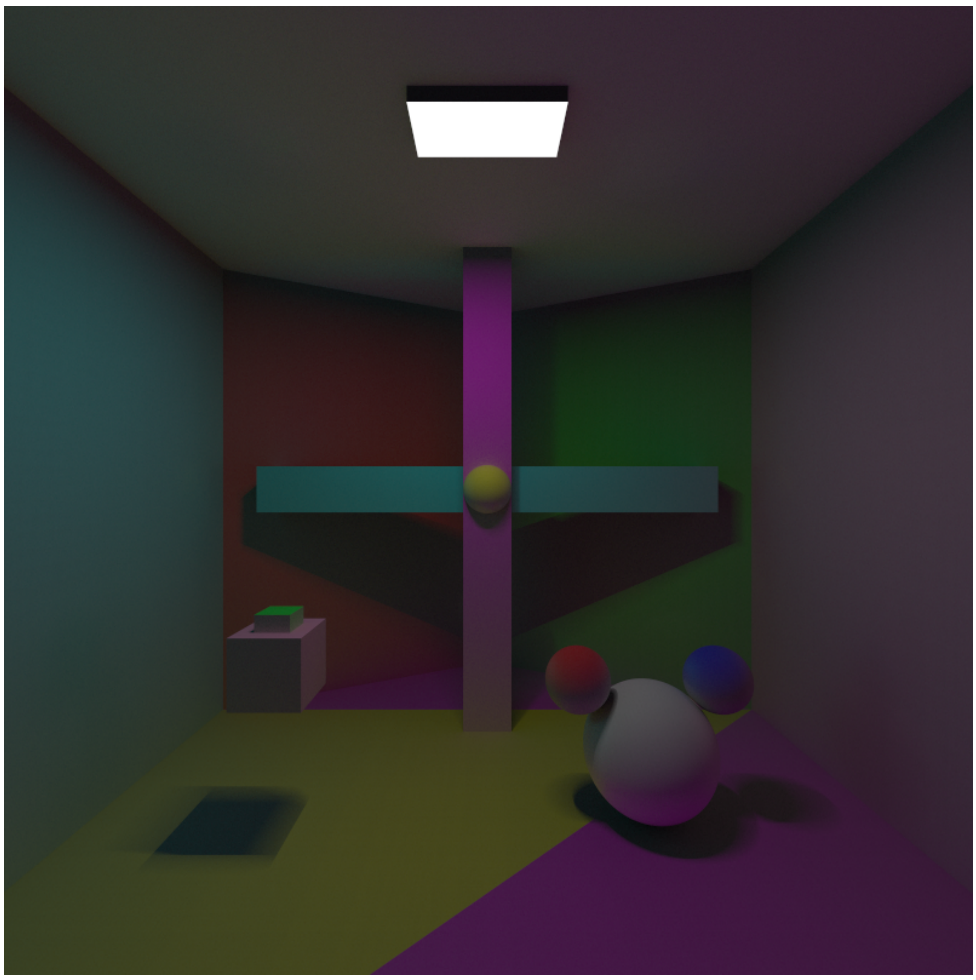


Figure 4.2: A rendered scene with 800x800 pixels of only diffuse surfaces, using 256 ray samples per pixel. The image was rendered with amplified contribution from indirect.

Chapter 5

Discussion

The implemented Monte Carlo ray tracer produced results that demonstrated ideal reflections, diffuse reflections and refractions in a good manner. A few interesting observations of the results are highlighted in this chapter.

Figure 4.1 can be used to study the glass material. The right glass sphere is completely transparent and only takes the color of the surrounding scene. The middle sphere has a light blue glass material and looks dark blue. However, looking at the middle sphere in the reflection of the mirror sphere, the blue color almost disappears completely. How much the glass color affects its appearance seems to be correlated with how much light that enters the sphere opposite the viewing direction, as the dark blue tint only is visible when the opposite side is in the shadow of the mirror sphere. The left sphere shows much more of its purple color in both directions, but also has a substantially larger color vector. Furthermore, a distinct line can be seen in the spheres to the left and right. This line is the result of the maximum angle θ_{max}, see equation 3.5. The effects of equation 3.6 can also be seen as faint reflections of the light source can be seen in all glass spheres.

Studying Figure 4.1a, that only uses one ray per pixel, the image appears grainy and full of artifacts. The pepper noise might be caused by diffuse reflections hitting several shadows in a row. The salt noise presumably comes from diffuse bounces hitting a light source within the first few bounces. The shadows appear very sharp, since there's only one shadow ray sent per light source. Should more shadow rays be produced, the shadows would become smoother. However, the same effect can be achieved with supersampling. The supersampling will smooth out the image and in turn produce a better result, as seen in Figure 4.1, where the number of samples goes from 1 to 256. As mentioned before, no shadow rays were sent to light sources. It would lead to shadow rays being sent from a light source to itself. One could exclude light sources from receiving shadow rays, but they should not appear darker when not under direct illumination.

The grainy salt noise underneath the glass spheres forms low level caustics when the number of samples are high enough. The caustics are seen on the diffuse boxes and on the floor beneath. Caustics are formed when light gets concentrated through reflection of highly glossy surfaces or refraction through transparent objects. A way to improve the caustics would be the use of a photon map. The light sources will send out a large number of photons toward the glass and mirror objects, with the photon map storing the values. The high density of photons will create more realistic caustic patterns, as they aren't estimated like the rest of the scene and have a higher resolution. Another photon map could also be used for the remaining scene. By sending out radiance and saving the values the number of diffuse bounces could be reduced by reusing the saved data, thereby shortening computing time.

Studying Figure 4.2 the color bleeding between diffuse surfaces appears very evidently. The bottom of the purple column receives much influence from the yellow floor, since the column is very thin and the floor very wide. Thus, there's a large chance of the rays bouncing from the column to the floor, receiving the floor's color. Up on the roof, color bleeding from the floor in the middle is seen, while the sides take

after the walls. There is even coloration from the cyan column. The reason for the very eminent color bleeding is probably the result from the roof not receiving any direct illumination and is therefore only lit up by indirect illumination. The white sphere placed on the ground has a soft line between the yellow and purple, being indirect illumination from the floor appearing very realistic. The indirect illumination in Figure 4.2 was boosted, to better illustrate the color bleeding between diffuse surfaces, as seen in the difference between that image and Figure 4.1.

The intersection point might not be fully correct, as they are not computed with infinite accuracy. This will sometimes lead to shadow and reflected rays being spawned within an object. A ray with the origin on the inside results in the ray intersecting with the target it spawned from, making it bounce around inside the object. Should the ray be a shadow ray, it would produce a shadow even though the object is under direct illumination. This was countered with an offset of the ray's origin, which worked well by adding 4% of the normal vector.

Bibliography

[1] Anon. *Snell's Law*. LibreTexts. URL: https://eng.libretexts.org/@go/page/346 (visited on 10/28/2021).

[2] Scratchapixel. *Ray Tracing: Rendering a Triangle Möller-Trumbore algorithm*. URL: https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-rendering-a-triangle/moller-trumbore-ray-triangle-intersection (visited on 10/28/2021).

[3] Softschools. *Inverse Square Law Formula Inverse Square Law Formula*. URL: https://www.softschools.com/formulas/physics/inverse_square_law_formula/82 (visited on 10/28/2021).

[4] Elizabeth Stapel. *Cramer's rule Cramer's rule*. URL: https://www.purplemath.com/modules/cramers.htm (visited on 10/29/2021).

[5] László Szirmay-Kalos. "Monte-Carlo Methods In Global Illumination". In: (Jan. 2000).