Rendering Global Illumination

Emma Forsling Parborg
Martin Gråd

TNCG15 - Advanced Global Illumination and Rendering
Linköping University

January 12, 2015

Abstract

Global illumination is a set a different techniques which can be used to render more realistic 3D scenes than local lighting models. In this article, a number of different illumination methods are briefly presented, followed by a more thorough description of Monte Carlo ray tracing and its implementation. The algorithm is implemented in the classic Cornell Box scene, consisting of diffuse walls as well as transparent and intransparent objects. The project aims to achieve a realistic illumination in this scene.

The results for the project illustrate that many important aspects of global illumination are implemented. Light interactions such as reflections and refractions of light rays are successfully integrated into the algorithm. Furthermore, effects such as color bleeding and caustics can be seen in the rendered images, albeit only in small amounts. The fundamentals can be said to be in place, while some details still could benefit from some improvement. This, along with ideas for future studies is discussed in the report.

CONTENTS

LIST OF FIGURES

CHAPTER

1

INTRODUCTION

Global illumination is a name for a number of techniques which seek to create more realistic renderings of 3D scenes than traditional local lighting models. Local models take into account only interactions between individual points and one or more light sources. In the real world, light bounces between surfaces, making objects illuminate each other. Global illumination methods seek to model such behavior by for each point computing lighting contributions from other points in the scene, in addition to those directly from the light sources.

There are many different models for rendering global lighting, including ray tracing, radiosity and photon mapping. These methods are presented and briefly discussed below.

1.1 Ray tracing

Ray tracing is a technique for rendering images by mapping three-dimensional scenes onto a two-dimensional view plane, representing pixels on the screen (Wikipedia 2014d). This is achieved by tracing rays representing the light, backwards through the scene. The rays start at a defined eye position, go through the pixels on the view plane, and continue into the scene, see Figure 1.1.

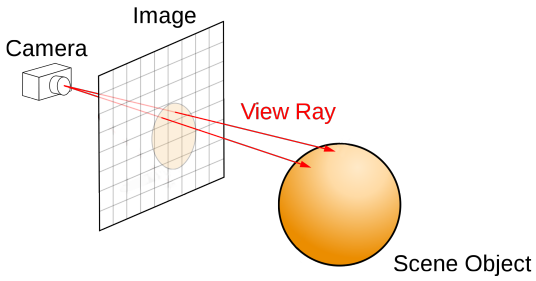


Figure 1.1: Representation of ray tracing, showing a scene object and the camera, consisting of a viewer position and an image plane (based on Wikipedia 2008).

Once they hit a surface, the color value at that point is used to derive a value for the current pixel. There are different ray tracing techniques, of which Whitted ray tracing and Monte Carlo ray tracing will be presented here.

1.1.1 Whitted ray tracing

Whitted ray tracing introduces light interactions between objects, making this technique a global illumination method (Wikipedia 2014d). These interactions consist of reflections and refractions of light, and are computed using the material properties of intersected objects. Reflections and refractions generate new rays, which are traced further into the scene and contribute to the color of the pixel that spawned the original ray. At each intersection point, a ray can spawn one or two new rays: only a reflected ray, only a refracted ray, or both. Which rays are created is determined by the material properties of the intersected object, as well as the direction of the incoming ray. All objects in the scene need to be defined as transparent, non-transparent or diffuse. Transparent objects can spawn both reflected and refracted rays, while non-transparent objects generate only a reflected ray. Diffuse objects spawn no new rays.

In addition to reflected and refracted rays, so called shadow rays are generated at each intersection point, Figure 1.2. These rays are used to determine if the current point is in shadow from each light source. They are defined as going from the point on the surface and toward the light source. If this ray intersects any other object between its starting point and its end point, the point in question is in shadow; if there is no intersection, the point is in direct light. This information is used to determine if the current point will contribute to illumination of the pixel that is being evaluated.

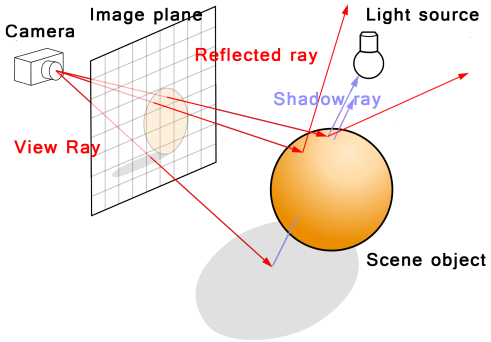


Figure 1.2: Representation of Whitted ray tracing (based on Wikipedia 2008).

1.1.2 Monte Carlo ray tracing

Monte Carlo ray tracing, or path tracing, builds upon the Whitted ray tracing scheme by introducing a way to compute other light interactions than only perfect reflections and refractions (Wann Jensen 2001, Wikipedia 2014b). This is done by randomly sampling possible light directions at each point. By describing material properties of surfaces by bidirectional reflectance distribution functions, or BRDF:s, Monte Carlo ray tracing can render global illumination of a wider variety of scenes. A BRDF expresses the ratio of reflected radiance to incoming irradiance at a certain point, and can be used to model arbitrary materials for use in the ray tracing algorithm. An important aspect of this is that it enables rendering of indirect illumination. This makes Monte Carlo ray tracing a more general method than Whitted ray tracing.

Thanks to the random sampling of light directions, the results from Monte Carlo ray tracing will converge on a correct solution of the rendering equation with an increasing number of samples, making this method a photorealistic algorithm. Averaging an increasing number of renders will produce the same results, which means that Monte carlo ray tracing is also an *unbiased* algorithm.

1.2 Radiosity

Radiosity is a technique for solving the rendering equation for surfaces that reflect light diffusely (Wann Jensen 2001, Wikipedia 2014c). The name of the method comes from the radiometric quantity that is calculated. With the Radiosity technique the surfaces of the scene are divided into small patches. For each pair of patches a view factor is computed. The view factor describes the relative orientation within each pair, and is in turn used to calculate the mutual illumination for

the patches. If a patch lies between a pair, the view factor is set to zero.

The radiosity method starts at the light sources, and iteratively transfers radiosity to the patches until all light has been distributed in the scene. For each iteration the light is absorbed, and the radiosity that each patch sends out is a sum of the self-emitted radiosity and the incoming radiosity from all other patches. The radiosity method takes time to calculate, and for practicality it can be done in separate steps in which each iteration can be rendered to continuously verify the results.

The Radiosity method is not dependent on where the camera is in the scene. When the computations are done the camera can be moved freely within the scene, without the need for recomputations. Radiosity computes lighting distributions quite efficiently for scenes with diffuse materials and simple objects. In more complex scenes, however, the number of patches needs to be increased which in turn increases the required number of computations.

1.3 Hybrid and multi-pass techniques

Ray tracing algorithms excel at rendering specular surfaces, but falter when it comes to diffuse materials. Whitted ray tracing lacks support for these kinds of materials, and while Monte Carlo ray tracing does support it, its efficiency is far behind that of other methods that exclusively render diffuse lighting, such as radiosity. For this reason, hybrid and multi-pass techniques have been developed (Wann Jensen 2001). These techniques combine different methods to handle various aspects of a scene. For example, Radiosity can be used as a first pass to render diffuse light interactions, and a ray tracer used as a second pass to visualize the specular lighting within the scene.

1.4 Photon mapping

Photon mapping is a method to render global illumination using a different approach than ray tracing (Wann Jensen 1996, 1998, 2001). It was developed with the goal to allow for more efficient rendering of scenes than what pure Monte Carlo ray tracing does, while still retaining the advantages of this method. Photon mapping differs from previously mentioned rendering schemes in that it stores lighting information in a separate independent data structure, a photon map, instead of storing it within the scene geometry.

A photon map consists of photons, which are small light packages that have been emitted and traced through a scene. The photons represent the lighting within the scene and are used to efficiently render the model. The photons are emitted from the light sources in the scene, and intersect objects in the same way as rays do in ray tracing. Similarly, when a photon hits a surface, it can be reflected or refracted and continue within the scene, or be terminated. Information about these photon hits is what is stored in the photon map.

Photon mapping is a two-pass method, which means that the algorithm consists of two separate, major parts. The first pass is Photon tracing, during which the photon map is constructed. In the second pass, Rendering, the scene is rendered using the information that is stored in the photon map. This is achieved through computing a radiance estimate based on photon density and incident direction for each point in the scene. When rendering a scene, different kinds of photon maps can be used, depending on the required application. The algorithm can simulate caustics, diffuse inter-reflection and subsurface scattering, and this can be utilized by adapting the model to suit current needs. The photon mapping method approximately solves the rendering equation, and is a *biased* rendering algorithm in that the only way to produce a correct solution is to use a large number of photons; as opposed to for example Monte Carlo ray tracing, averaging an increasing number of renders does not converge on a correct solution.

1.5 Aim

The aim of this project is to implement a well-proven method to render global illumination. For this, Monte Carlo ray tracing has been chosen.

1.6 Structure of the report

The first section is *Introduction*, in which different global lighting models have been briefly discussed. In the second section, *Background*, the scene setup will be explained, along with important mathematical concepts that the rendering algorithm is based on. The implementation of the chosen global lighting model will be described in greater detail in the third section, *Method*. The fourth section is *Results and benchmarks*, in which different sets of parameters and their influence on the results and performance will be studied. The fifth and last section is *Discussion*, where the results will be analyzed.

CHAPTER

2

BACKGROUND

In this section, the scene setup will be explained, along with important mathematical concepts that the rendering algorithm is based on.

2.1 Setting up the scene

The scene consists of the objects that are to be rendered, i.e. a *Cornell box*, and a camera that comprises a viewer position as well as an image plane.

2.1.1 The Cornell box

The scene to be rendered in this project is a *Cornell box*, which is a common test scene for rendering algorithms (Goral et al. 1984). The model consists of five walls forming a room with an opening facing the viewer, one or more light sources, and a number of different objects placed in the room. In this project, there is one area light source, placed in the ceiling, two spheres with different optical properties and a cube; see Figure 2.1.

2.1.2 The camera

The camera consists of a viewer position, a view direction and an image plane. The image plane is made up of pixels covering its entire area, which represent the

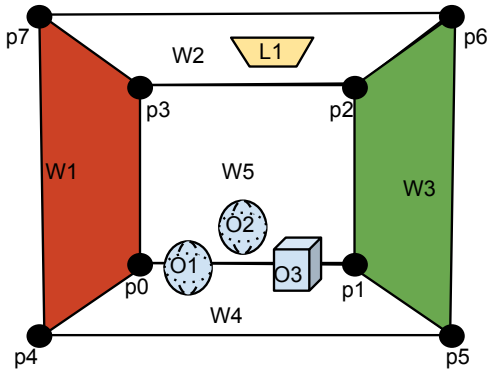


Figure 2.1: The Cornell box scene

image that is to be rendered. The viewer position is set at an arbitrary distance from the center of the back wall, *W5* in Figure 2.1, and the camera faces inward, with a view direction that is parallel to the remaining walls, W1-W4, see Figure 2.2. The image plane has a certain size, and from using this together with the viewer position, the focal length of the camera can be determined. This makes it possible to have the generated image fit the scene in that it lets it fill up the entire image without excluding anything, regardless of viewer position. Changing the viewer position changes only the focal length. Moving the viewer closer to the scene corresponds to making the angle of view wider; moving it farther away makes it narrower.

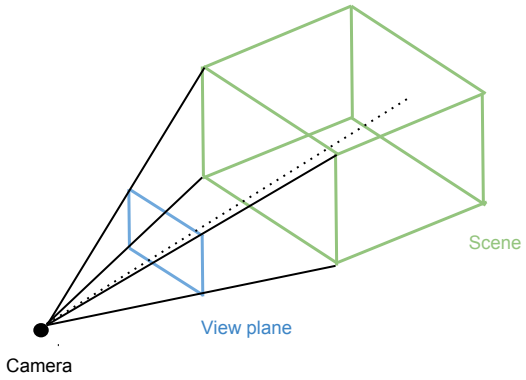


Figure 2.2: The camera setup, illustrating camera alignment with the scene as well as view plane positioning.

2.2 Defining rays

The representation of light as rays is based on the fact that photons move along straight lines in empty space, and that radiance is constant along such paths. Furthermore, the photons scatter symmetrically when intersecting surfaces, which means that a ray path is the same from the incident and exitant directions (Wann Jensen 2001).

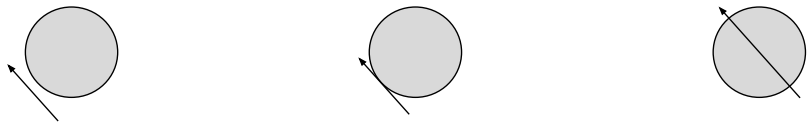
2.3 Calculating ray intersections

To determine ray intersections with objects, different computations are made depending on what kind of object the rays intersect. The different scenarios for intersection are: ray-sphere intersection, ray-cube intersection and ray-room intersection. All objects in the scene except for the sphere consist of rectangles. The cube has six sides; the room has five.

2.3.1 Ray-sphere intersection

The computation of ray-sphere intersections is based on the mathematical solution of line-sphere intersections (AmBrSoft 2012a), extended to account for ray type and the fact that light moves in one direction only. A line can have zero, one or

two intersections with a sphere, as illustrated in Figure 2.3. In the case of two intersections, the algorithm needs to consider if the ray is inside or outside the current object. If the ray is inside an object, the correct intersection point is the one that is farthest from the ray origin; if it is outside, the correct intersection is given by the closest point.



(a) No intersection. (b) Single Intersection. (c) Two intersections.

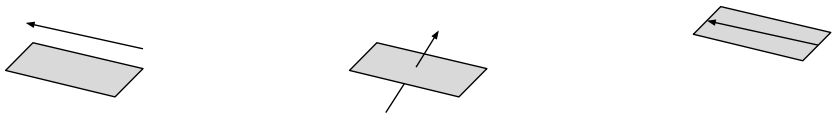
Figure 2.3: Determine ray intersection with a Sphere

2.3.2 Ray-cube and ray-room intersections

For computing intersections with a cube or the room, the same algorithm is used; the only difference is the number of sides to loop through. The algorithm is based on line-plane intersection (AmBrSoft 2012b), Figure 2.4, extended to account for the distance from a ray origin to each rectangle, as well as whether the ray is inside or outside the current object. Each side of the room or a cube is a rectangle, for which an intersection point can be calculated using the mentioned mathematical model. The algorithm also needs to determine if the plane-line intersection that has been computed using this method is in fact contained within the current rectangle. As in ray-sphere intersections, a line can intersect a cube at zero, one or two points. Likewise, the correct intersection point is determined using information about whether or not the current ray is inside the object. For the room, only zero or one intersection can occur, since it has an opening facing the camera.

2.3.3 Light source intersection

When a ray intersects the light source, the radiance from the light is fed directly into the pixel from which the ray originated. Akin to the room and the cube, the light source is represented by a rectangle, thus enabling the use of the same algorithm as for these objects to determine if there is such an intersection.



(a) No intersection. (b) Single intersection. (c) Infinite no. of intersections.

Figure 2.4: Determine ray intersection with a Plane.

2.4 Tracing rays

In order to achieve global illumination, emitted rays need to bounce around and acquire lighting contributions from multiple points within the scene. Once a ray intersection has been determined, a number of new rays are created, starting at this position. The first of these is a *shadow ray*, which is generated for all types of surfaces. In addition to this ray, reflected and refracted rays can be spawned for specular objects in the scene.

2.4.1 Shadow rays

From each point in the scene that has been intersected by a ray, a shadow ray is sent toward the lights source. This is done to determine if the current point is in shadow or direct light. By utilizing the same algorithm as for any other ray in the scene, the point closest to the ray origin in the direction toward the light can be found. If this point is the starting point itself or on the light source, there is no occlusion; if it is any other point, there current point is occluded.

2.4.2 Reflections and refractions

For specular surfaces, each intersected point can spawn reflected or refracted rays, depending on the material of the object and the angle between the incident light and normal for the object. Figure 2.5 illustrates reflection and refraction when a light ray comes from one medium and hits another. The incident ray creates an angle, θ_1, with the normal. For perfect reflection this angle is equal to the angle between the reflected ray and the normal, θ_1'. Refracted rays are created from refraction. Refraction is the change in direction caused by the change in speed that occurs when light travels from one medium to another. The refracted ray creates an angle, θ_2, with the negative normal. The refractive indices n_1 and

n_2, are dimensionless constants which describes how light, or any other radiation, propagates through that medium. Air and glass have refractive indices of approximately 1.0 and 1.5, respectively, and these have been used in this project.

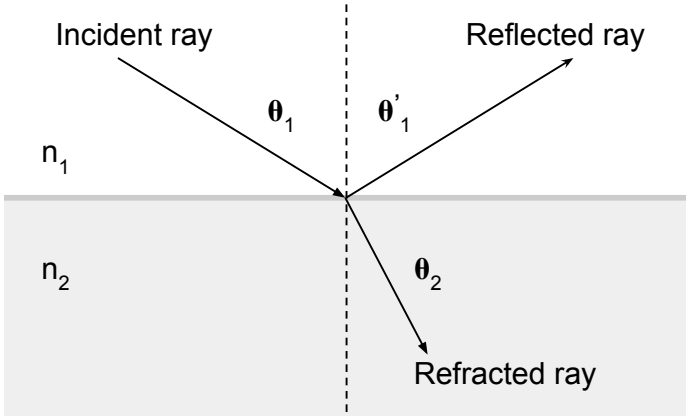


Figure 2.5: Reflection and refraction with two different media.

In vector form, the reflected and refracted rays can be derived without computing the sine values and angles (Wikipedia 2014e). By avoiding such computations the algorithm is more effective. To determine the reflected ray the following equations are used:

$$\cos\theta_1 = -\mathbf{l} \cdot \mathbf{n} \tag{2.1}$$

$$\mathbf{v}_{reflect} = \mathbf{l} + 2\cos\theta_1 \mathbf{n} \tag{2.2}$$

$\mathbf{l}$ is the incident ray direction vector, $\mathbf{n}$ is the normal vector pointing out from the surface that is intersected and $\mathbf{v}_{reflect}$ is the perfect reflection ray vector. When $\cos\theta_1$ is determined, $\cos\theta_2$ is calculated by using Snell's law, Eq. 2.3, in combination with the Pythagorean identity. The derived formula for $\cos\theta_2$ is displayed in Eq. 2.4.

$$sin\theta_2 = \left(\frac{n_1}{n_2}\right)\sin\theta_1 \tag{2.3}$$

$$\cos\theta_2 = \sqrt{1 - \left(\frac{n_1}{n_2}\right)^2 (1 - \cos^2\theta_1)} \tag{2.4}$$

These equations explain the relationship between the angles for the incoming ray and the refracted ray. With this relationship in mind, the refracted ray vector is calculated with the following equations:

$$\mathbf{v}_{refract} = \left(\frac{n_1}{n_2}\right)\mathbf{l} + \left(\frac{n_1}{n_2}\cos\theta_1 - \cos\theta_2\right) \tag{2.5}$$

The radiance that flows through each ray is decided with following equation:

$$R = \left(\frac{n_1 - n_2}{n_1 + n_2}\right) \tag{2.6}$$

$$T = 1 - R \tag{2.7}$$

Where R and T represents the radiance in the reflected ray and the refractive ray, respectively.

2.4.3 Monte Carlo and hemispherical sampling

To render light interplay for diffuse surfaces, a uniform sampling of random directions is performed. For lambertian surfaces, light is scattered in all possible directions. To achieve this, each ray that intersects such an object spawns a new ray, with a random direction. The possible directions are limited to a hemisphere centered around the intersection point.

2.4.4 Ray termination using Russian roulette

The method Russian Roulette is used to determine if a ray should be terminated or reflected (Wann Jensen 2002). The method is based on the fact that the chance of a material reflecting or absorbing a ray differs between materials. Diffuse materials have an increased chance of absorbing a ray, while specular materials have a higher probability of reflecting it. The first step of Russian roulette is to set a value, α, for the chance of absorption, ranging from 0 to 1. A random number is then generated, in the same range. If $1 - \alpha$ is less than the random number that is generated, the ray should be absorbed and thereby terminated. The next step in Russian Roulette is to set a maximum depth, and for every iteration check that it is not exceeded. If it is, the ray should be terminated.

2.5 Calculating local lighting contribution

At each point in the scene where a ray intersects an object, a local lighting contribution is calculated and added to the current pixel value. This contribution is derived using *Lambertian reflectance* (Wikipedia 2014a), according to the following equation:

$$I_d = N \cdot L * C * I_L \tag{2.8}$$

where N is the surface normal, L is the direction to the light source, C is the surface color and I_d is the light source intensity. This equation holds for $N \cdot L = [0, 1]$, since this dot product equals the cosine of the angle between the vectors, according to the following equation:

$$N \cdot L = |N||L| \cos\theta \tag{2.9}$$

Values outside of this range correspond to surfaces that face away from the light source, which should have no local lighting contribution.

2.5.1 Converting radiance to RGB

To convert the radiance to RGB, the resulting colors are converted to range between 0 and 255. Before this, a gamma correction is applied to deliberately distort the image so that when distortion occurs again, when it is displayed on a screen, the image will display correct brightness values (Wikipedia 2015). The equation used for gamma correction is:

$$\mathbf{v}_c = \mathbf{v}_s^{1/\gamma} \tag{2.10}$$

Where $\mathbf{v}_c$ is the result of gamma correction and $\mathbf{v}_s$ is the computed radiance value, which has to be between 0 and 1. To ensure this, a clamp function is used to set values that are greater or smaller to one and zero, respectively. The γ used in this project is 2.2. After gamma correction is performed the following equation is used to convert the gamma corrected radiance $\mathbf{v}_c$ to RGB values:

$$\mathbf{c}_{rgb} = (255 * \mathbf{v}_c) \tag{2.11}$$

CHAPTER

3

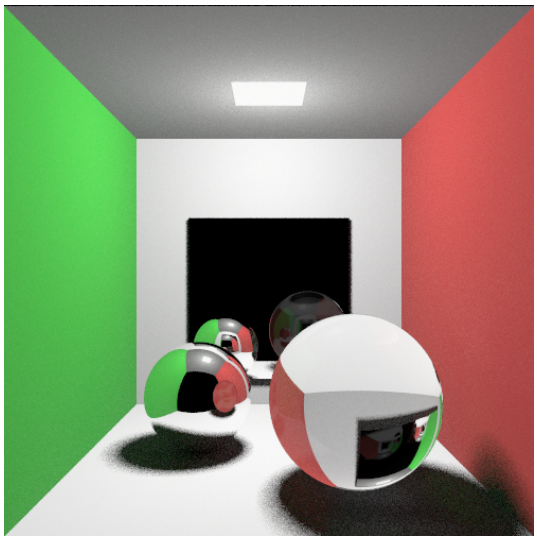
RESULTS AND BENCHMARKS

The results for this project are divided into a few sections, in which different aspects are presented. The transparent object in the scene is represented by a sphere to the right, and the intransparent objects are represented by a sphere to the left and a cube in the background, constituting a mirror on the wall. The project is implemented in C++ and the image that is derived is saved to ppm-format. Attempts to parallelize the code were made in this project, using OpenMP, but the rendered images did not portray the same results as whose rendered without it, see Appendix A.2.

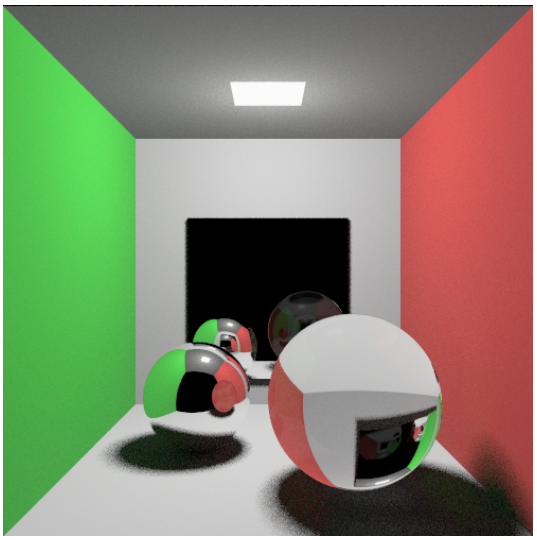
3.1 Results with different background colors

The following images represent the scene with different background colors for the floor, ceiling and the back wall. The images have been rendered with 32 rays per pixel and the resolution 500×500 pixels. This comparison is made to illustrate the difference in color bleeding effects that occurred in the project. In the image with white background, Figure 3.1a, the color bleeding effect is not visible, except for in the shadows of the spheres. In the images with gray through black backgrounds the color bleeding effect is increasing, see Figure 3.1b, 3.1c, 3.1d. In the last image the caustics on the floor from the transparent sphere is the most apparent. In Appendix A.1, the image with the black background can be viewed

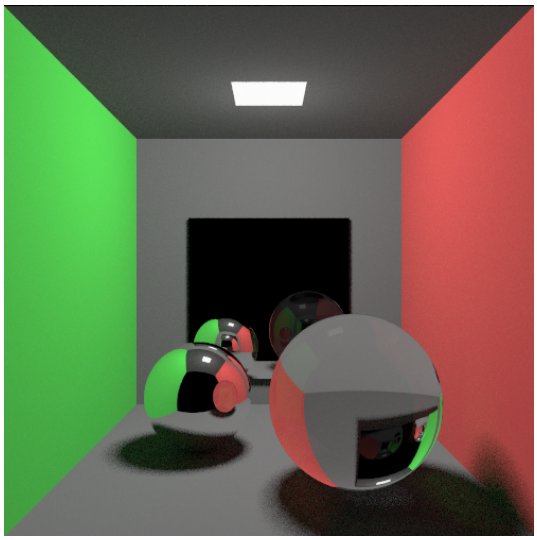
with image resolution 1000 × 1000, rendered with 64 rays per pixel.



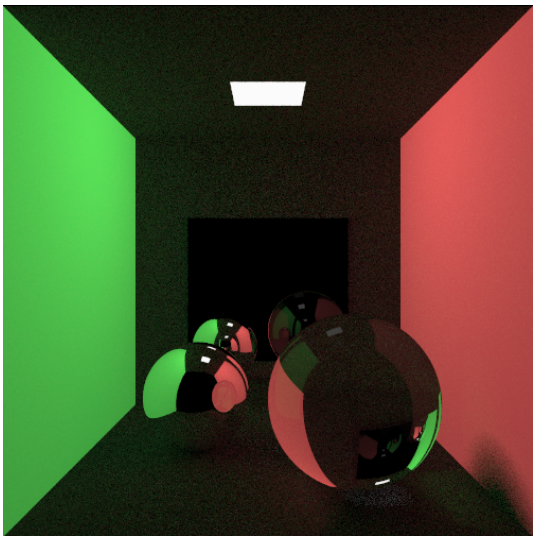
(a) White background



(b) Bright gray background



(c) Dark gray background

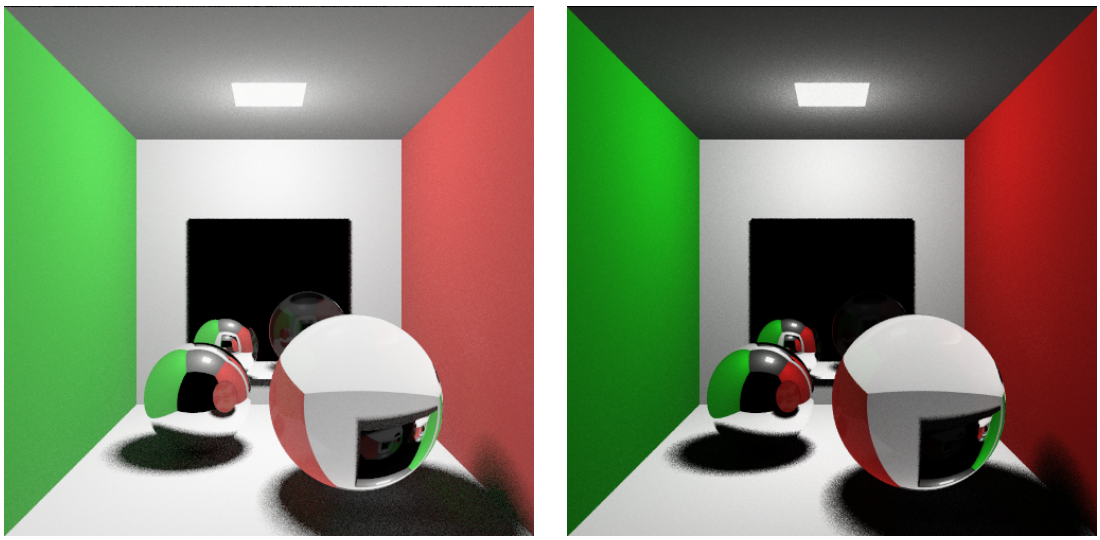


(d) Black background

Figure 3.1: Images with different background colors.

3.2 Results with and without gamma correction

The following images represent the scene rendered with and without a gamma correction of 2.2. The images have been rendered with 32 rays per pixel and the resolution 500 × 500 pixels. In the image with gamma correction, Figure 3.2a, the reflections of both of the spheres are visible in the cube in the background. However, in the image without gamma correction, Figure 3.2b, the reflection of the transparent sphere is less visible. This image also contains darker and sharper shadows, with less color bleeding effects.



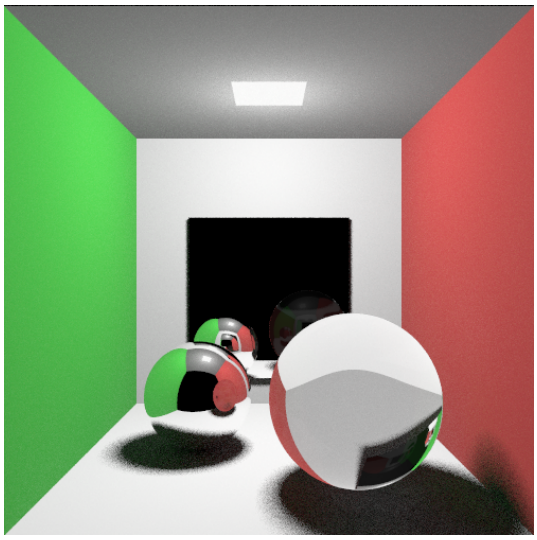
(a) With gamma correction

(b) Without gamma correction.

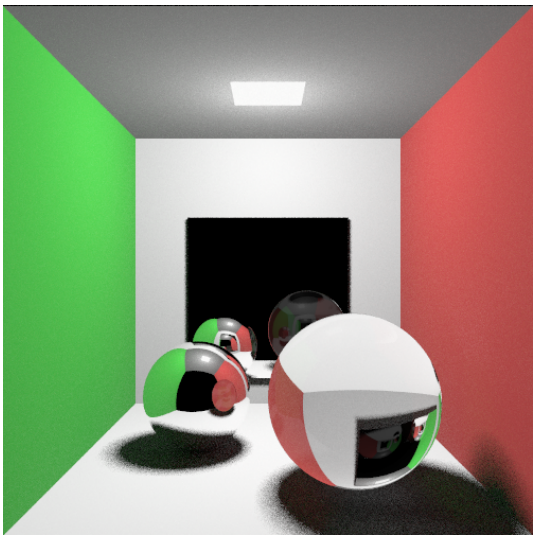
Figure 3.2: Images illustrating with and without gamma correction

3.3 Results when varying the refractive indices

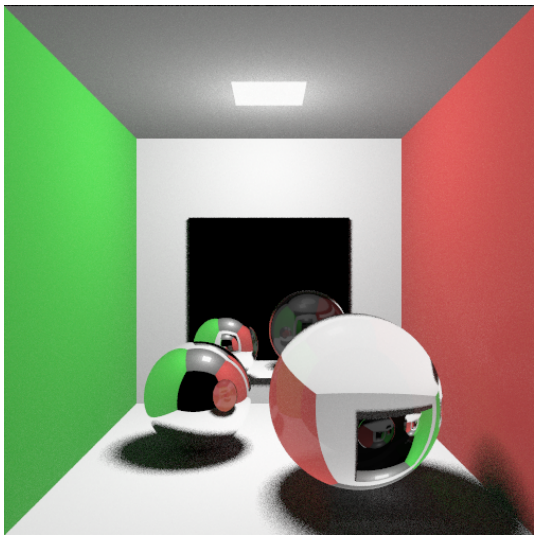
The following images represent a scene in which the refractive index for the transparent sphere has been changed. The images have been rendered with 32 rays per pixel and the resolution 500 × 500 pixels. As illustrated in the images, the greater the index is, the larger the refraction angle is. A similar change in ratio between the reflection and refraction also presents itself.



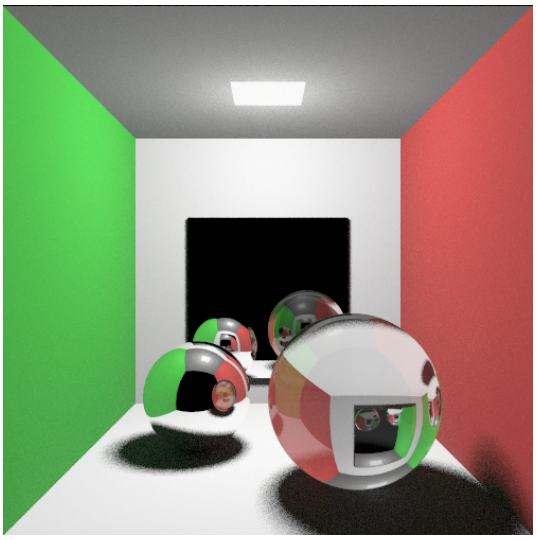
(a) Refractive index of 1.3



(b) Refractive index of 1.5



(c) Refractive index of 1.7



(d) Refractive index of 2.5

Figure 3.3: Images with varying refractive index for the transparent sphere.

CHAPTER

4

DISCUSSION

4.1 Analyzing results

The results show that a number of important aspects of global illumination are implemented in the project. Most of these can be considered successful, while the implementation of some of them seems slightly flawed.

4.1.1 Reflections and refractions

Reflections and refractions of light enable objects to affect each other and be visible in multiple places within the scene. In the figures presented, the small sphere can be seen reflected on the cube in the back. The same reflection can also be seen indirectly through the larger, transparent sphere. This sphere can similarly be seen reflected on the cube, as well as on the smaller sphere. These effects can be considered correct.

4.1.2 Color bleeding

Another aspect of global illumination is color bleeding, which is the way in which surfaces illuminate each other, spreading their colors onto other surfaces. The underlying procedures for this effect are in place, but the implementation seems to be partly deficient. The color bleeding effect is most easily discernible in Figure

3.1d, where the green and red walls can be seen coloring the black surfaces on the floor, ceiling and back wall. However, this effect fades with increased brightness on those surfaces, which is shown in Figures 3.1a - 3.1c. Color bleeding should still be visible on brighter surfaces, which leads to the conclusion that some part of the implementation is incorrect. One theory is that some color information goes missing either in adding color contributions together or in saving the image. On bright surfaces, a higher value in a certain color channel might be discarded if all values happen to be fully saturated, i.e. larger than one.

4.1.3 Caustics

Another important effect which is made possible by the Monte Carlo method is caustics. Similarly to the color bleeding, this effect can be seen in Figure 3.1d, but not as clearly in figures 3.1a - 3.1c. This is also likely due to some oversight in the implementation, since the fundamental mechanics are in place.

4.2 Improvements and future studies

One improvement that can be made for this projects is to make the code more efficient by succesfully parallelizing it. The attempts made in this project were to parallelize it for each pixel, but these attempts were unfortunately unsuccessful. Another improvement that can be performed in this project concerns performance and execution of the program using very high pixel and image resolutions. During the last phase of the project, when the hemispherical sampling was introduced, a memory leak that previously was undetected was discovered, for which a solution not has been found. Lastly, concerning the caustics and the color bleeding effects, as mentioned earlier, improvements on this is something that should be made.

CHAPTER

5

CONCLUSION

This project aimed to created a Cornell Box scene in which the global illumination technique Monte Carlo ray tracing was to be implemented. The result for this project is a scene containing some global illumination aspects such as reflections, refractions, color bleeding and caustics. The first two have been successfully implemented in this project, while the other two only partially work. In images with dark background color these effects are visible, but not as clearly in images with bright background color.

REFERENCES

AmBrSoft. (2012a). *Intersection of a Line and a Sphere*. http://www.ambrsoft.com/TrigoCalc/Spher/SpherLineIntersection_.htm 2014-11-17.

AmBrSoft. (2012b). *Intersections of a Plane and a Line*. http://www.ambrsoft.com/TrigoCalc/Plan3D/PlaneLineIntersection_.htm 2014-11-17

Goral C.M., Torrance, K.E., Greenberg D.P. & Battaile B. (1984). Modeling the Interaction of Light Between Diffuse Surfaces. *Siggraph Computer Graphics*. Vol. 18, No. 3.

Wann Jensen, H. (1996). Global Illumination using Photon Maps. *Rendering Techniques*. pages 21–30.

Wann Jensen, H. & Christensen P. H. (1998). Efficient simulation of light transport in scenes with participating media using photon maps. *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. pages 311-320.

Wann Jensen, H. (2001) *Realistic Image Synthesis Using Photon Mapping*. A K Peters, Ltd.

Wann Jensen, H. (2002).*A Practical Guide to Global Illumination using Photon Mapping*. http://www.cs.princeton.edu/courses/archive/fall02/cs526/papers/course43sig02.pdf 2015-12-01

Wikipedia. (2008). *Ray Trace Diagram.* http://en.wikipedia.org/wiki/Ray_tracing_(graphics)#mediaviewer/File:Ray_trace_diagram.svg, 2014-11-14

Wikipedia. (2014a). http://en.wikipedia.org/wiki/Lambertian_reflectance 2015-01-11

Wikipedia. (2014b). *Path tracing.* http://en.wikipedia.org/wiki/Path_tracing 2014-11-17

Wikipedia. (2014c) *Radiosity (computer graphics).* http://en.wikipedia.org/wiki/Radiosity_(computer_graphics) 2014-11-17

Wikipedia. (2014d). *Ray tracing.* http://en.wikipedia.org/wiki/Ray_tracing_(graphics) 2014-11-17

Wikipedia. (2014e). *Snell's law.* http://en.wikipedia.org/wiki/Snell%27s_law 2015-01-11

Wikipedia. (2015) *Gamma correction.* http://en.wikipedia.org/wiki/Gamma_correction 2015-01-11

APPENDIX

A

IMAGES

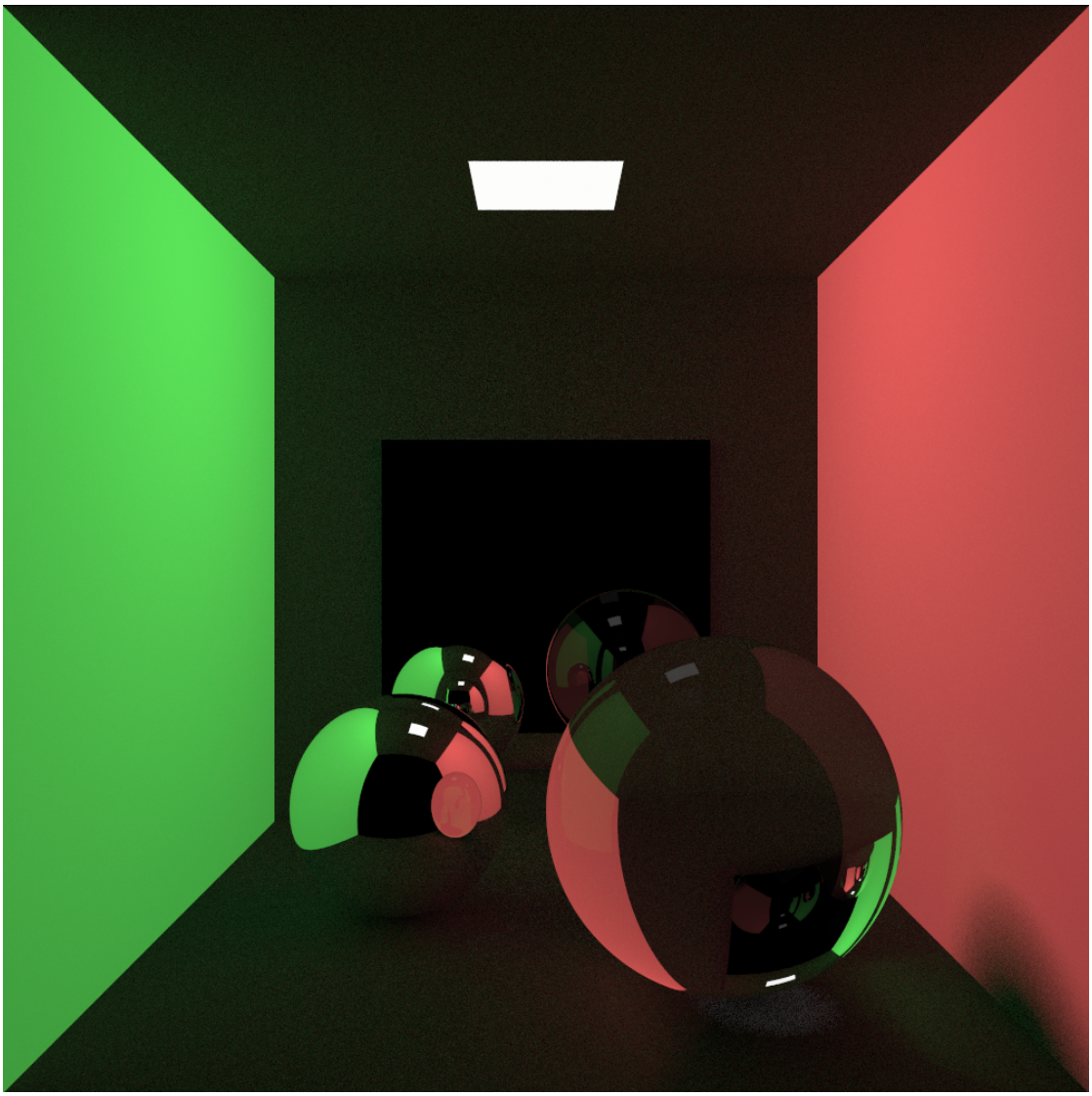


Figure A.1: A larger image of the Cornell Box scene, in which black has been used to enhance the color bleeding effect in the room and the caustics from the transparent sphere. The image resolution is set to 1000 $\times$ 1000 and the pixel resolution is set to 64 rays per pixel

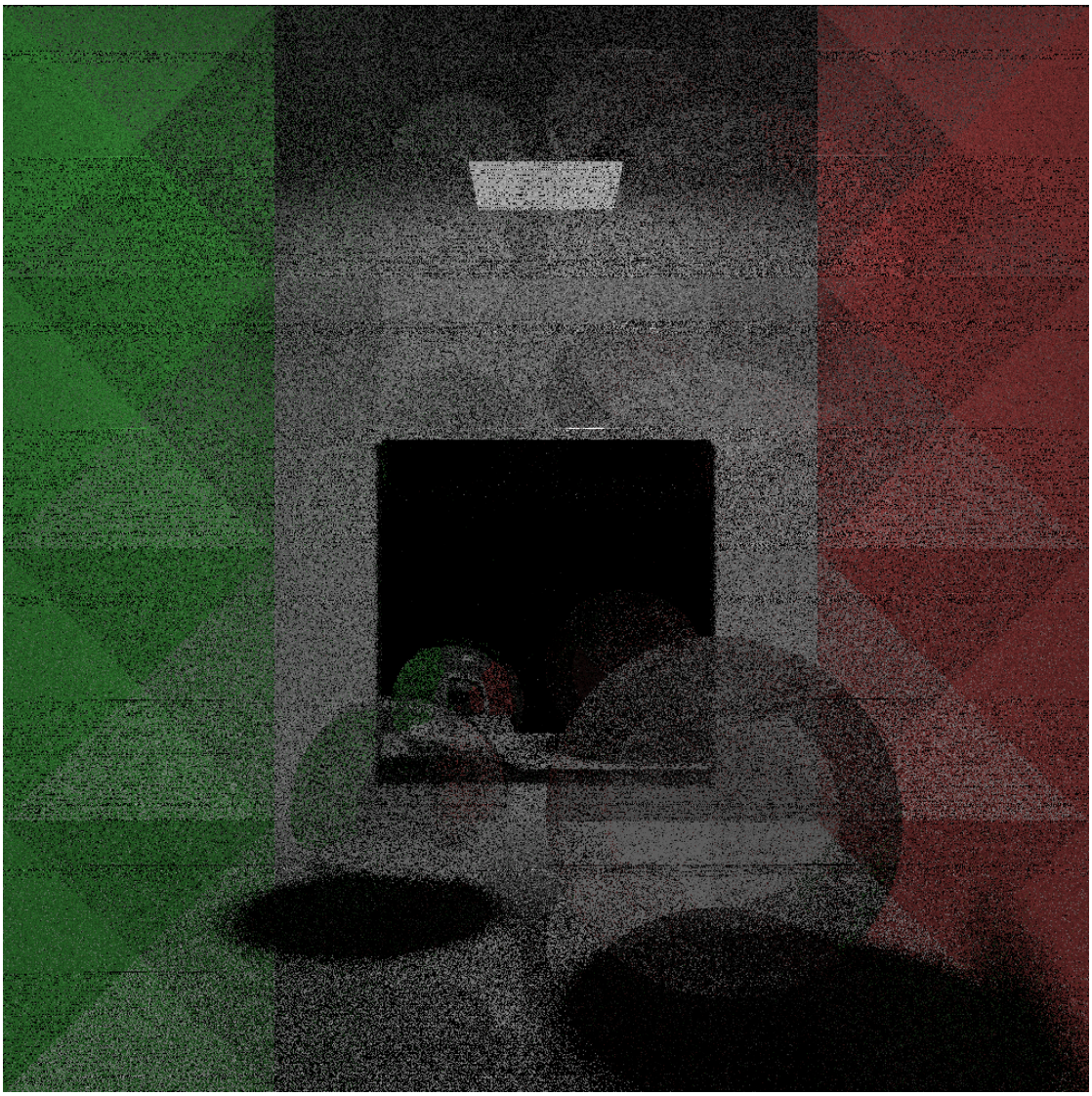


Figure A.2: Example of the results achieved when trying to implement OpenMP to have the program execute on multiple cores.