Monte Carlo Ray Tracer
Advanced Global Illumination and Rendering
TNCG15

Karin Reidarman, karre663
Caroline Gard, carga635

December 2017
Linköping University

Abstract

The paper describes in detail the computational algorithms of the Global Illumination method Monte Carlo Ray Tracing. The paper also describes a Monte Carlo renderer developed for a project which was implemented for a scene including Lambertian surfaces. The report includes results from the implemented renderer and a discussion about quality improvements, advantages and disadvantages with the algorithm. A Monte Carlo Ray Tracer was implemented using the programming language C++ and the project was done in the course Global Illumination and Rendering (TNCG15) at Linköping University.

1 Introduction

This section presents a wider description of different methods used for global illumination as they have been presented in their original papers and why they are important to global illumination.

1.1 Global Lightning Model

Global Illumination is a general term of a group of different algorithms and methods for achieving more photorealistic renderings in 3D computer graphics. Other effects such as shadows and caustics, which appears due to indirect lightning, are also included in the topic global illumination. Earlier developed rendering techniques in 3D computer graphics, only consider the direct light contribution from the light source when computing the reflected rays. The limitation of such methods is that scenes rendered using only direct illumination will significantly impair in realism in the rendered image. Therefore, to achieve global illumination, not only the direct illumination should be considered, but also the indirect illumination, which is light coming from the light source reflected onto surfaces in the scene [5]. In addition to reflected light, examples of typical properties global illumination also consider are refracted light and shadow rays.

For rendered images with global illumination, light that has been reflected multiple times must necessarily be considered in the algorithm. To find the color of an object point is a complex problem since the color depends on the reflected light rays and other object nearby. If only direct illumination adds to the algorithm, the only information captured is the surface and the associated color hit by the light ray, which will set the color of the pixel. The disadvantage with only using direct illumination, is the loss of light and color information from objects in the visual scene. In real-world scenes, the objects around the point of interest, will affect the color and appearance of the point, not only the direct light and the color from the surface of the hit point. For example, the color of a point on a white surface will get affected by the green wall next to the point. This is due to the reflected light rays, with source from the light that is reflected on the green wall to bring color information to the point of interest.

To achieve more photorealistic images, shadows are also one important addition to the process. When calculating every point in the scene, a determination regarding if the current point is located in shadow or not needs to be made. That is, to see if there are any objects between the point of interest and the light sources in the scene, to determine if the point should be illuminated by the light source or not.

1.2 Whitted Ray Tracing

To solve the global illumination problem, one commonly used method is the Whitted Ray Tracing algorithm. Basic ray tracing is used to determine the color information for every pixel in the image plane without consider reflected rays. In further developed methods for ray tracing, such as Whitted Ray Tracing, additional computations are added in the process.

Whitted Ray Tracing is a rendering technique within global illumination, used for generating images with higher quality of photorealism. The resulting image from Whitted Ray Tracing is generated by tracing a ray from the camera and generating a path in which the ray travels through the 3D scene. Depending on the material properties of the intersection point and light contribution from the light sources, the reflected ray will contribute differently to the resulting color. In Whitted Ray Tracing each ray can possibly result in three new rays at the intersection point; reflected rays, refracted rays and shadow rays. If a ray gets reflected or refracted depends on the material of the surface, see image 1. A specular surface generates a reflected ray, with the same outgoing angle from the normal to the surface as the incoming angle. This reflection results in a mirror-like reflections on the surface. Transparent materials generates a refracted ray which is reflected through the material. The angle of the refracted ray depends on the material properties from and to the ray travels, and can be calculated using Snell's Law [6]. In addition to reflected and refracted rays, shadow rays are also introduced in the Whitted Ray Tracing algorithm. For every intersection point, a shadow ray is casted from the intersection point towards the light source. The shadow ray are traced to determine if any objects is positioned between the light source and the intersection point, to define if the should be illuminated or not.

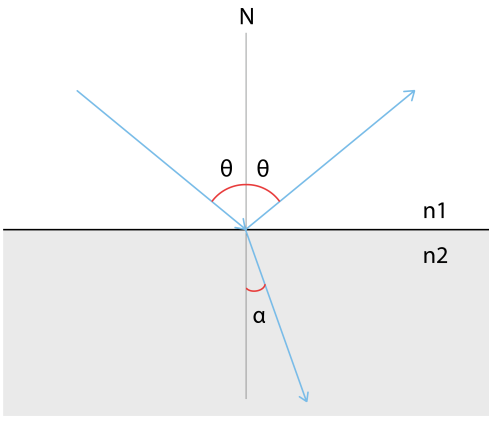

Figure 1: A ray that is being reflected and refracted.

Whitted Ray Tracing renders an image with a high degree of photoreal-

ism but at the cost of computation time, which makes the technique suitable for static images instead of moving shots. The method handles effects such as shadows, mirror-like objects and transparent objects, and was a great success when if first was developed. Compared to modern methods for global illumination, Whitted Ray Tracing is not fully photorealistic, since it lacks physical effects such as soft shadows, glossy reflection and caustics [6]. One solution is a combination with other methods to receive an better level of realism.

1.3 Radiosity

Radiosity is a global illumination model that is used to generate images that realistically simulate the scene by taking into account the light when it propagates through the scene. The light travels from a light source and is reflected on a surface in the scene, and continue to propagate until the energy is transformed into light heat for example.

The radiosity method is based on Heat Transfer Theory, which describes radiation as the transfer of energy leaving a surface when it has been thermally excited. The theory can be applied to describe the light energy transferred from one surface to another. The light emitted from a surface patch is the sum of the light energy emitted by the surface itself and the energy the surface reflects. The observer only perceives the total amount of light coming in and cannot tell the two components apart. So the computation of radiosity is a description of light from a particular patch reaching every other patch in the scene, causing the ambient lighting in the scene.

The method assumes that all surfaces are perfectly diffuse reflectors [2], which means that the light intersected with the surface is spread equally in all directions from the intersection point. See figure 2. The advantage is that the surface intensities are independent of view direction and therefore the scene only needs to be rendered once [3]. Then the camera can be moved around without re-rendering the scene, making it a common used rendering technique in the architecture industry.

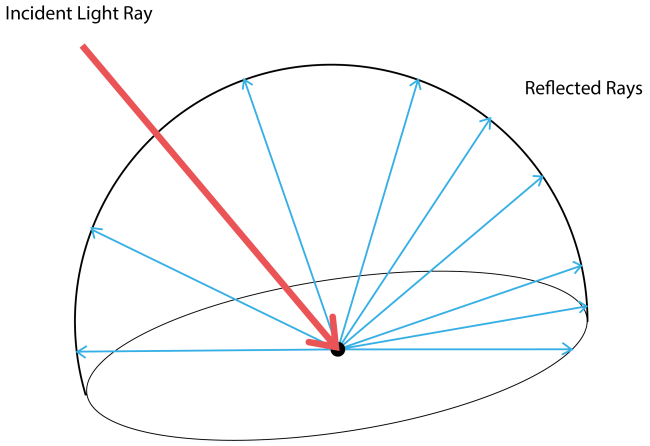

Figure 2: A ray that hits a diffuse surface will spread equally in all directions within the hemisphere.

1.4 Monte Carlo Ray Tracing

Monte Carlo Ray Tracing is an extended and further advanced version of the Ray Tracing algorithm. The difference is that the method also relies on random sampling, which introduces soft shadows, an effect that solves problems of aliasing artifacts which may occur in the regular Ray Tracing algorithms. The Monte Carlo Ray Tracing method is a combination of several advanced methods and algorithms to generate photorealism in rendered images. In general, the method uses ray tracing and repeated randomness for sampling to produce desired result.

When computing the ray path of reflected rays, Monte Carlo ray tracing uses the method Russian Roulette as a stopping criteria to determine if the ray at the intersection point should terminate or get further reflected after the current intersection [8]. The decision is based on coincidence and is completely random.

1.5 Photon Mapping

Photon Mapping is a two-pass global illumination method used to render images to achieve photorealistic results [7]. Photon mapping is one of the latest developed and most commonly used algorithms for global illumination, since the algorithm generates accurate results with lower cost of computation time compared to other methods. The method is similar to other methods for global illumination, such as Monte Carlo Ray Tracing and Ray Tracing, but with some advantages. Instead of computations using light rays, the method uses photons. A photon can be described as a massless particle storing elements of light such as the radiosity and the incoming direction of light. The information can later be integrated with any BRDF, *Bidirectional Reflectance Distribution Function*, to approximate the intensity of the light in the point of interest [4].

The method includes two passes where the first pass is a forward photon trace from the light source, and the second pass is a backward photon tracing from the camera where the value of radiance is estimated. In the first pass, a photon map is created when photons are individually traced when they interact with objects in the scene. In the second pass, the photon map created in the first pass, is used to compute indirect illumination including caustics and shadows. The second pass is possible computed using the Monte Carlo Ray Tracing algorithm.

1.6 Structure of The Paper

For the following sections in the paper, techniques regarding Monte Carlo Ray tracing is discussed. In section two, the Monte Carlo techniques implemented in the projects is presented. Section three, demonstrates obtained results and benchmarks. Discussion and outlook is found in section four that also includes possible future developments for the method.

2 Background

This section presents Monte Carlo Ray Tracing techniques and algorithms implemented in the project, and describes equations regarding ray tracing, reflections and light contribution. The implemented project included a Monte Carlo Ray Tracer that models intransparent surfaces using Lambertian reflection.

2.1 The Rendering Equation

The Monte Carlo Ray Tracing algorithm is developed to solve the rendering equation, see equation 1 below. This equation is further used to compute the radiance leaving the surface and to compute the color of the pixel.

$$L(x \to \omega_{out}) = L_e(x \to \omega_{out}) + \int f_r(x, \omega_{in}, \omega_o ut) L(x \leftarrow \omega_{in}) \cos\theta_{in} \mathrm{d}\omega_{in} \quad (1)$$

where L is the total radiance leaving the surface in direction ω_{out}, L_e describes the light source contribution, f_r is the BRDF, L is the spectral radiance. BRDF defines the amount of light that is reflected from a surface. Radiance is the radiant flux of a surface and is determined by the angle of the incoming or outgoing light direction. The equation for BRDF is seen in equation 2.

$$f_r = \frac{p}{\pi} \quad (2)$$

The algorithm work in the way that rays are sent through the pixel plane out in the scene, and are traced when getting reflected along the way. To calculate the color of the pixel using the rendering equation, the computation is done backwards using the importance. This is discussed more in detail later in this chapter.

2.2 The Scene

The scene in the project is a hexagonal room and consists of a roof, a floor and six walls. The floor and roof are white, and each wall have one of the colors; orange, red, blue, green, yellow or purple. The walls, floor and ceiling are constructed of triangles added to a triangle mesh. In addition, the scene contains two objects, a sphere and a tetrahedron. The tetrahedron is constructed of four triangle put together. All objects in the scene have the surface property of Lambertian surfaces.

2.3 The Camera and Pixel Plane

The camera is positioned at equal space from the ceiling and floor, in one end of the room, looking at the other end of the room. The pixel plane is positioned in front of the camera, see image 3.

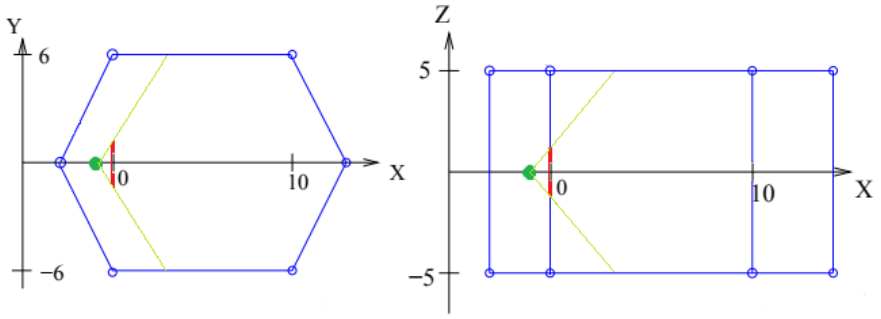

Figure 3: The green marker is the camera and the green lines are the camera frustum. The red line is the pixel plane. The left image is seen from above and the right is seen from the side.

2.4 Surface Intersections

When rays are sent out in the scene through the pixel plane, an intersection will certainly occur since the room in the scene is closed. For every ray in the scene, the Möller-Trumbore algorithm is used to identify all the triangles that the ray in question intersects. Every triangle in the scene consist of three corner points, and barycentric coordinates (u, v) with $(u, v) \in [0, 1]$ are used to describe every point on the current triangle, given by the equation 3 below.

$$T(u, v) = (1 - u - v)\mathbf{v}_0 + u\mathbf{v}_1 + v\mathbf{v}_2 \tag{3}$$

The current ray is described by the equation 4

$$\mathbf{x}(t) = \mathbf{p}_s - t(\mathbf{p}_e - \mathbf{p}_s) \tag{4}$$

where $\mathbf{p}_s$ is the starting point of the ray, $\mathbf{p}_e$ is the end point of the ray and t describes the path along the ray and is within the interval of zero and one. When combining the two equations, the equations are rewritten to the equation 5

$$u\mathbf{E}_1 + v\mathbf{E}_2 - t\mathbf{D} = \mathbf{T} \tag{5}$$

where $\mathbf{E}_1$ and $\mathbf{E}_2$ describes two edges of the triangle, $\mathbf{D}$ is the tracked ray and $\mathbf{T}$ is the intersected point on the triangle.

To find an intersection between a triangle in the scene and the ray, it is important to consider that several intersection may occur for one light ray. In that case, the closest intersection is of interest since the other points will be placed behind that object. Therefore, the Möller-Trumbore algorithm must be computed for every triangle in the triangle mesh to find the nearest intersection point in the scene for the current ray. For every triangle intersection, the distance from the current point and the intersected object is saved. When all

intersections have been recorded the one with the shortest distance is chosen as the intersection that we want to look at.

Since the scene does not completely consist of triangles, but also a sphere, every ray also needs to be checked if it will intersect the sphere. If there is an intersection the distance from the origin to the intersection is calculated and checked against the closest triangle that was recorded from the triangle intersection test. If the sphere intersection distance is smaller than the triangle intersection distance, the sphere intersection point will be recorded in the ray path as the hit point and the other intersections will be disregarded. If the sphere intersection distance is larger that the triangle intersection distance, the triangle intersection will be kept as the hit point.

The method for testing if a ray intersects a sphere is similar to the method used for the triangles. First the distance between the centre of the sphere and the ray is calculated. If the distance is larger than the radius of the sphere there is no intersection at all. If the distance is smaller there are two possibilities. Either there is one intersection or two intersections, see figure 4 [1].

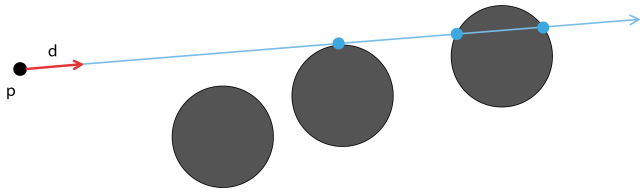

Figure 4: Three possibilities of sphere intersection.

2.5 Reflection of Light

There are different types surfaces in a real-world scene, something that is important to recreate when generating global illumination. Different materials and textures will affect the behavior of the reflected light. Some surfaces absorbs parts of the light and reflect some of the light further, and the properties and direction of the reflected light will therefore depend on the properties of the surface hit by the light ray. Because this project only consider Lambertian surfaces, specular and transparent surfaces is not included in this section.

The project implemented reflection models for Lambertian surfaces, which describes diffuse and blurry reflections which absorbs part of the light when light rays get reflected. Unlike specular surfaces, where the law of reflection is used to reflect the light in the correct angle of intersection, the same method is not valid for Lambertian materials. Instead the angle of the reflected light is selected randomly for every intersection, which is described more in detail in section 2.6

below.

2.6 Radiance

In the scene, the radiance and the importance depend on each other. A radiance ray is sent out from the light source and intersects with an object in the scene, to then be reflected in a new direction. This will continue until the ray intersect with the camera in the scene, but not all rays emitted from the light source will intersect with the camera. To solve this problem, an importance ray is shot out from the camera to travel the same ray path as the importance ray but in the opposite direction. The importance value of the ray will attenuate as it is reflected on a surface, similar to the computed radiance loss when hitting an object.The factor with which they both decrease depends on the BRDF of the surface. For Lambertian reflections the BRDF is described as $f_r = \frac{p}{\pi}$ where p is the constant reflection coefficient in the interval of $[0, 1]$. If the importance going out from the camera in 5 is 1.0 and the BRDF at both point Y and X is 0.8, then the importance going out in direction ω_{out} is 0.64.

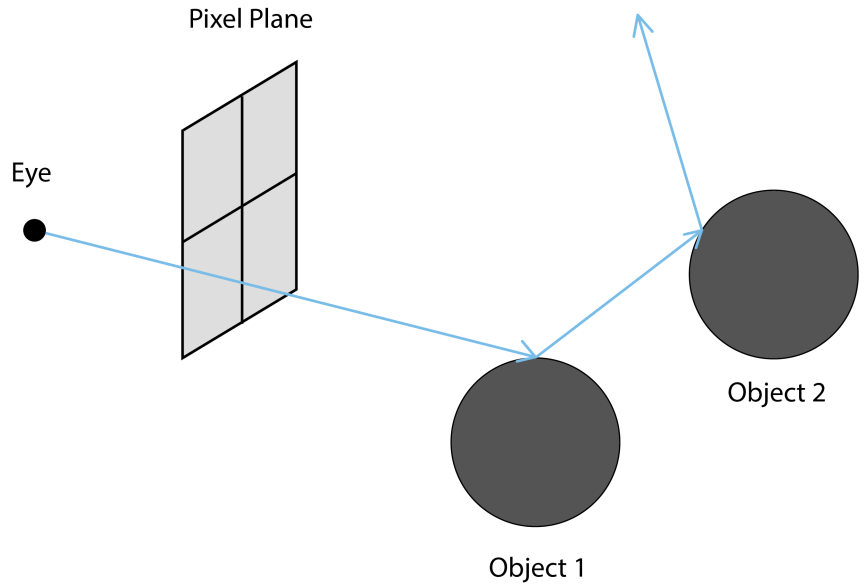

Figure 5: An importance ray path.

When the path from the camera to the light has been calculated, the radiance from each intersection point can be added to the total irradiance of the image pixel [3]. The radiance of one point, independent of the incoming importance ray, is calculated by summing the direct light contribution and the indirect light contribution. The direct light contribution is calculated by sending a ray from the point on the surface to the light source, assuming only one light source exist

since that is the case for the setup of the scene in the implemented project. If the ray does not hit another object on the path, the point is in direct light and the contribution Le is calculated according to equation (equ:dir$_l ight_c ont$).$L_e = lightIntensity * lightColor * k_d * \cos\theta$(6)where θ is the angle between the surface normal and the light ray (see figure 6), and k_d is a small diffuse reflection component.

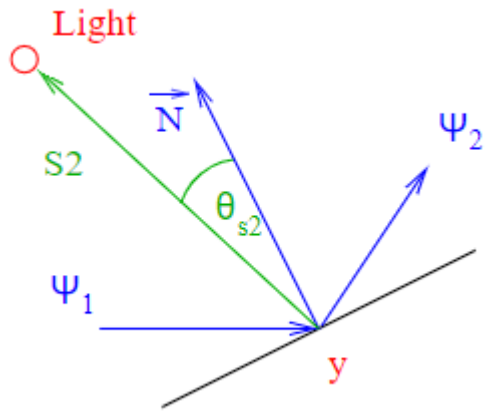

Figure 6: A ray s2 is cast from the surface point to the light.

If the point on the surface lies in the shadow of another object the direct light contribution will be zero.

The indirect light should be computed by calculating the integral over the hemisphere around the point and take into account all the light scattering that is coming into the hemisphere, that is essentially what the radiosity method computes, see equation 7. The hemisphere is a unit hemisphere that can be placed around the current point on the surface which the calculations are computed on The ray that goes through the middle of the hemisphere is pointing in the same direction as the normal of the surface.

$$\int f_r(x, \omega_{in}, \omega_o ut) L(x \leftarrow \omega_{in}) \cos\theta_{in} \mathrm{d}\omega_{in} \tag{7}$$

$$\langle F^N \rangle = \frac{1}{N} \sum_{i=0}^{N-1} \frac{f(X_i)}{pdf(X_i)} \tag{8}$$

However, in Monte Carlo, integration of the indirect light sampling is done by estimating the integral function 8. So rather than integrating the whole hemisphere, Monte Carlo take discrete samples by shooting out rays in different directions of the hemisphere and the color of the objects they hit are summed

and then divided by the number of sampled rays. In the implementation though, there is only one discrete sample ray that contributes to the indirect lighting, that ray is the next intersection point in the ray path.

The function to be sampled is the integral of the rendering equation 7. To sample the function, its Probability Distribution Function (PDF) (*Probability Distribution Function*) and the functions Cumulative Distribution Function (CDF) (*Cumulative Distribution Function*) is required to be solved [8]. One solution is that it can be concluded that the PDF is a constant, since all the surfaces are diffuse Lambertian in the scene, and so the distribution of the sample rays must be spread equally over the hemisphere. This results in that the PDF of the function is $\frac{1}{2\pi}$. The CDF can be seen as a function that returns the probability that some random variable X will be found to have a value equal or less than x.

The CDF helps to determine the random direction of the sample rays. If there is two random values $r_1, r_2 \in [0, 1]$ the CDF of the inclination(θ) and azimuth(ϕ) are:

$$\theta = \arccos 1 - r_1 \tag{9}$$

$$\phi = r_2 * 2\pi \tag{10}$$

When the color of a sample ray is summed to the indirect light contribution of the surface point, it is treated as a light source, meaning that the color needs to be multiplied with $\cos\theta$, where theta being the angle between the surface normal and the sample ray direction. Then as indicated in 8, the total indirect light needs to be divided by the number of sample rays multiplied by the PDF, see equation 11.

$$totalIndirectLight = \frac{totalIndirectLight}{N\frac{1}{2\pi}} \tag{11}$$

The total radiance of the surface point can then be calculated with equation 12.

$$totalRadiance = (directLight + indirectLight)(\frac{albedo}{\pi}) \tag{12}$$

2.7 Shadow Rays

Most rays sent out from the camera, will never intersect with any of the light sources. Despite this, points in the scene may be located in direct light. Shadow rays are introduced in the Monte Carlo algorithm to determine if the current point is located in shadow or in direct light, and introduce additional light channels to handle radiance at the intersection point. Shadow rays are created by casting rays from the intersection point towards a randomly generated point on the light source, see image 7. The idea is to determine if any objects is positioned between the intersection point and the light sources to compute the

distributed light contribution by tracing the ray. When using the Monte Carlo ray tracing method, several shadow rays are traced from the current point towards the light source. In the project the use of one, four or eight shadow rays were implemented for the same interaction point with randomly generated points on the light source. The technique results in soft shadow.

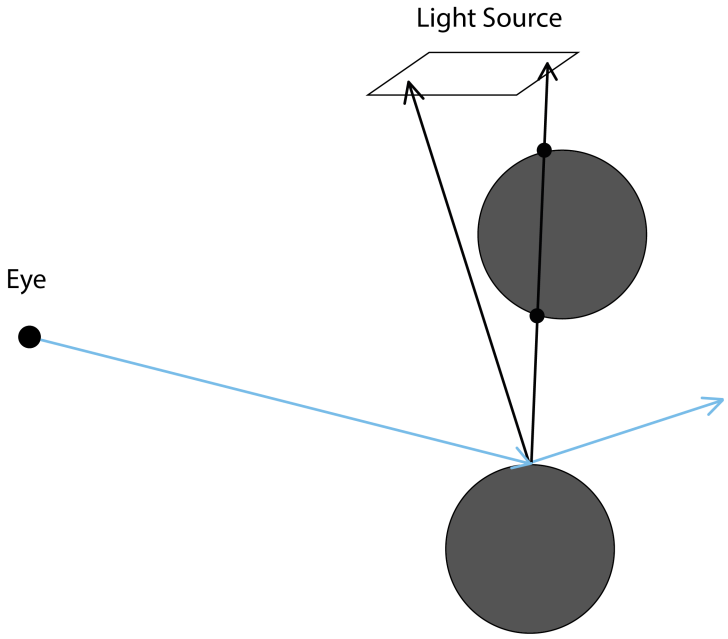

Figure 7: Soft shadowing

The next section present resulting images from the project using different resolutions and number of casted shadow rays. Additionally, the section lists benchmarks in the project as well as discussion about color bleeding and noise aliasing.

3 Results and Benchmarks

This section of the report present resulting images from the project using different resolutions and number of casted shadow rays. Additionally, the section lists benchmarks in the project as well as discussion about color bleeding and noise aliasing. 5 images have been rendered using varying values for number of rays sent out in the scene through each pixel, shadow rays per intersection point and number of allowed reflections. The benchmarks for each rendered image is recorded. The code is written in C++ and the computer used to do the rendering is a HP Z640 with the processor Intel(R) Xenon(R) CPU E5-1630 v3 @ 3.70GHz 3.70 GHz.

3.1 Color Bleeding

Radiance introduces color bleeding which is when objects in the scene affect the appearance of the objects nearby. This effect is caused by the reflected rays adding color information to the point of interest, additionally to the light contribution from the light source. The effect has an impact in global illumination and is required for achieving photorealistic results.

Image 9 is rendered with no further reflections after the first intersection between the light ray and an object in the scene. Image 8 is a rendered images with 4 rays in the ray path, 8 shadow rays and 8 rays per pixel. In the image the blue sphere is bleeding onto the white floor. Comparing the resulting images, the color bleeding only occur when a point gets lighting contribution from the indirect light, which does not happen in image 9. This is because if no rays are reflected, no color information will be added as indirect light to the point of interest, only color from the current point itself, and therefore no color bleeding will occur.

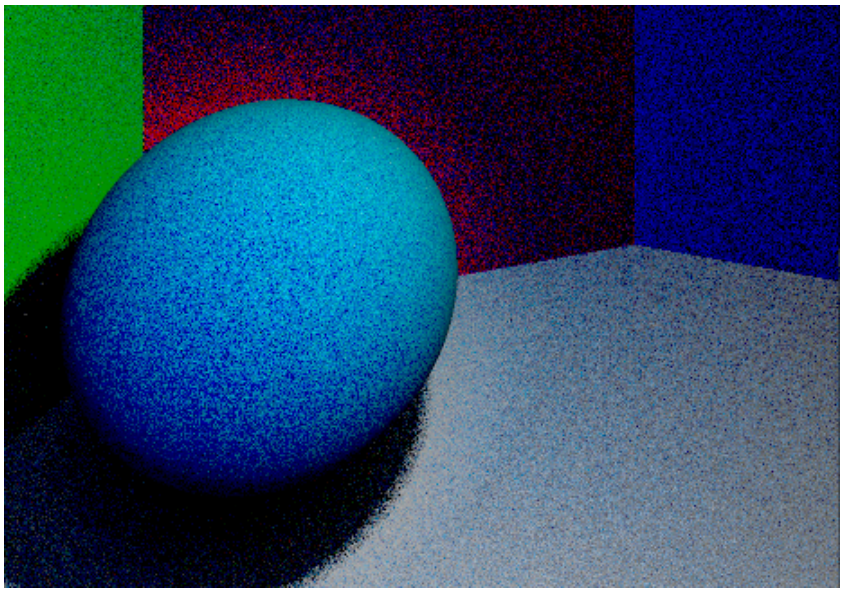

Figure 8: Color bleeding.

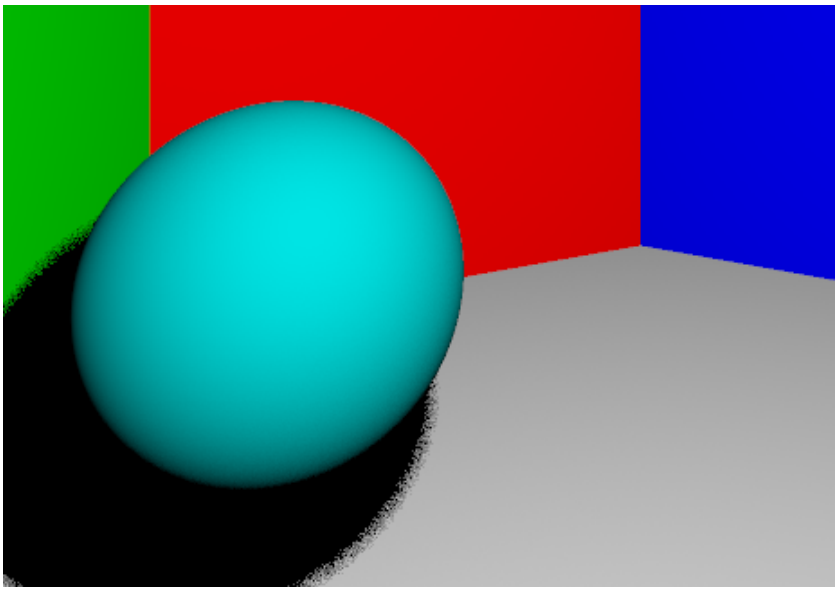

Figure 9: No color bleeding.

3.2 Noise and Aliasing Artifacts

Images rendered with ray tracing techniques tend to be noisy, a bottleneck within the ray tracing methods. Aliasing artifacts is also an effect that in some situations occurs when an image is undersampled, which generates pixelated edges, see image 10. The level of quality of the rendered image depends on the number of rays casted from the camera into the scene. Supersampling, or multisampling, is a spatial anti-aliasing method to improve the result of the rendered image and to reduce noise and aliasing artifacts. When using supersampling, instead of sampling one ray from the camera through each pixel out in the visual scene, several rays are casted through each pixel which improves the result significantly. When several rays are sampled for the same pixel, the average of the sampled colors, is calculated for the pixel color. The result of this approach is softened edges and smoothed noise in the rendered image, see image 11.

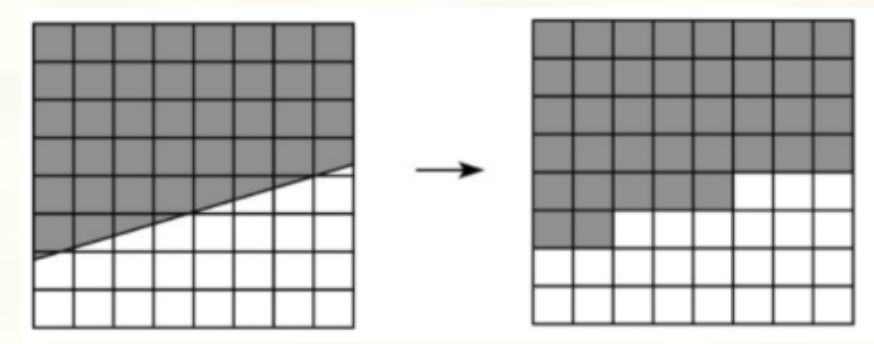

Figure 10: No supersampling

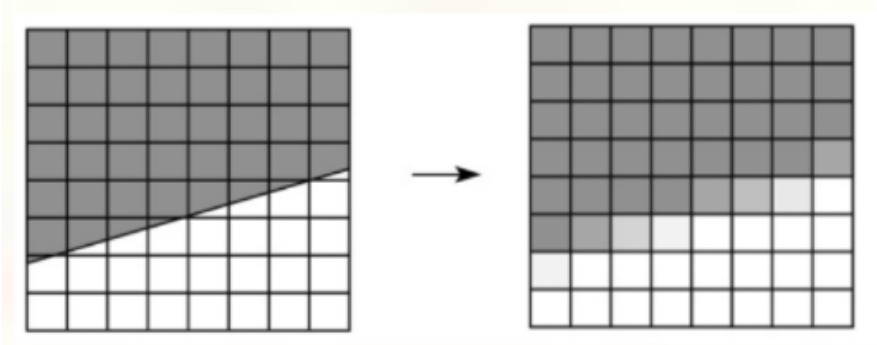

Figure 11: Supersampling

For Ray Tracing, the supersampling method is simply applied to the algo-

rithm, since each pixel in the renderer is computed separately and therefore additional rays, casted through each pixel can be simply added, see code 12. One possible improvement for supersampling is to randomize the position for the sampled rays within each pixel. Then there will not be a pattern created by the sample rays that is the same for all pixels.

One advantage with Monte Carlo Ray Tracing is that the method handles randomization which will prevent aliasing artifacts, which is a issue for some basic rendering methods, such as Whitted Ray Tracing. The Monte Carlo method uses the Probability Distribution Function to ensure that the direction of the sample rays is correctly casted in a direction to collect information that has significance.

The Monte Carlo Ray Tracing algorithm, uses the same technique to generate soft shadows similar to how the method generates soft edges of light distribution. From every intersection, between a ray and an object in the scene, shadow rays are sent out towards the light source and are tested if there is any object between the point of interest and the light source. The distribution of the shadow rays also depends on the PDF. However in the case for the light the PDF is $p(q) = \frac{1}{A}$, where q are the point on the light and A is the area of the light.

If an object is found when generating a shadow ray the point is considered to be located in shadow, and if not, the point receives light contribution from the direct light source. Again, an average from all the shadow rays will set the light contribution of the point in question. So, for the Monte Carlo algorithm, the point on the light source is randomly selected on the whole light area, with the help of the PDF, and so, also the shadow edges will be soft. In most other ray tracing techniques the light source is a point light and the shadows will be sharp.

```
for every pixel on the pixel plane {
    float fourth = 0.0005;
    glm::vec3 pos1(0, (y*0.002 - 0.999)+fourth, (z*0.002 - 0.999)+fourth);
    glm::vec3 pos2(0, (y*0.002 - 0.999)+fourth, (z*0.002 - 0.999)-fourth);
    glm::vec3 pos3(0, (y*0.002 - 0.999)-fourth, (z*0.002 - 0.999)+fourth);
    glm::vec3 pos4(0, (y*0.002 - 0.999)-fourth, (z*0.002 - 0.999)-fourth);

    Ray r1(eye, pos1);
    Ray r2(eye, pos2);
    Ray r3(eye, pos3);
    Ray r4(eye, pos4);

    color += scene.rayIntersection(r1);
    color += scene.rayIntersection(r2);
    color += scene.rayIntersection(r3);
    color += scene.rayIntersection(r4);
    color /= 4.0f;

    imagePlane[current pixel].setColor(color);
}
```

Figure 12: Generating 4 rays per pixel

3.3 Self Shadowing

When a ray is checked for intersection with objects in the scene, it may happen that the ray detects an intersection with the object which the ray originates from, which can possibly result in black pixels on the object. This is caused by some small numerical errors introduced by the fact that numbers in C++ can only be represented within a certain precision. Therefore, sometimes the intersection point is registered as being under the surface, see image 13.

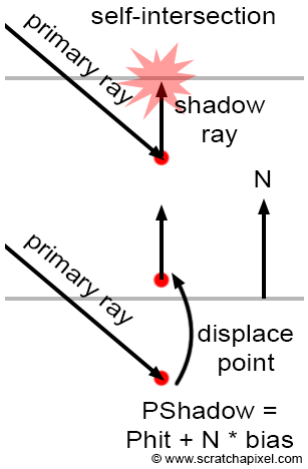

Figure 13: Intersection point is detected "under" the objects surface. Source: https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-shading/ligth-and-shadows

This error can be avoided by adding a margin to the intersection point. Instead of recording the intersection point as being right on the surface, the intersection point is moved up marginally along the surface normal. See figure 14

```
if (tryIntersectionObject(ray.normalized_direction,
                          ray.start_point,
                           a_scene_object, hit_point )){
    intersected_object.object = a_scene_object;
    intersected_object.hit_point = hit_point+ 0.001 * a_scene_object.normal;
    intersection_list.push_back(intersected_object);
}
```

Figure 14: Code to prevent self shadowing

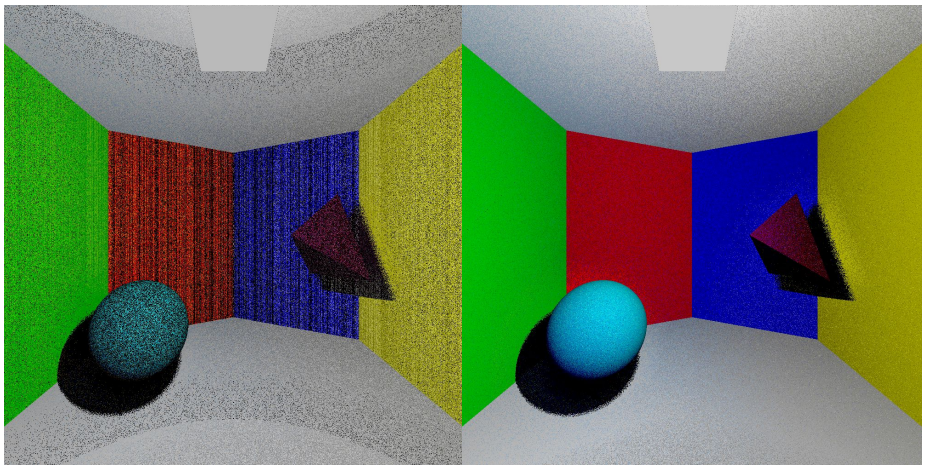

Figure 15: With self shadowing

Figure 16: No self shadowing

Image 15 is our scene with self shadowing. Image 16 is our scene without self shadowing.

3.4 Benchmarks

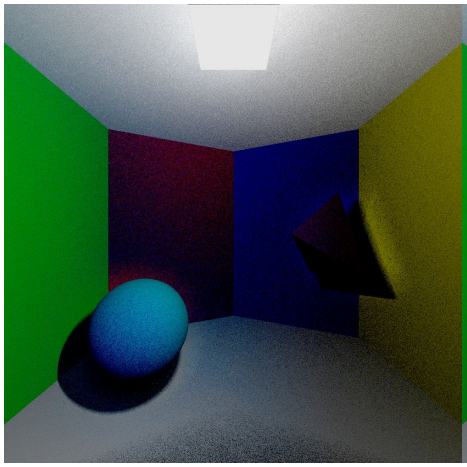

Figure 17: 3 reflected rays, 4 shadow rays and 4 rays per pixel

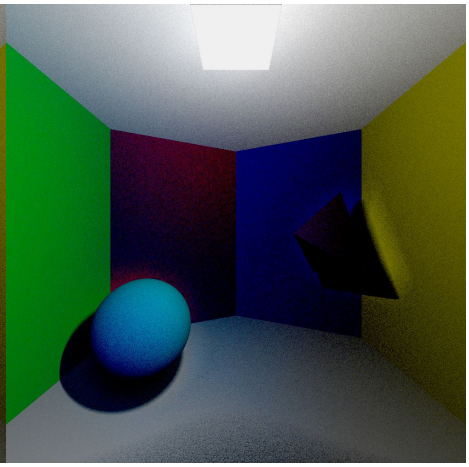

Figure 18: 3 reflected rays, 8 shadow rays and 8 rays per pixel

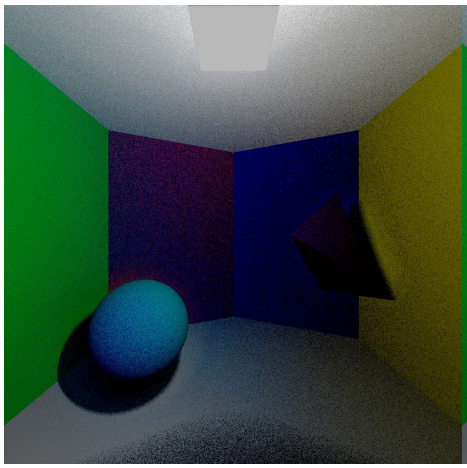

Figure 19: 8 reflected rays, 4 shadow rays and 8 rays per pixel

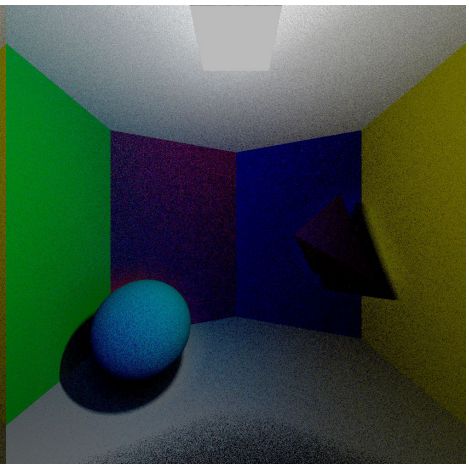

Figure 20: 8 reflected rays, 8 shadow rays and 4 rays per pixel

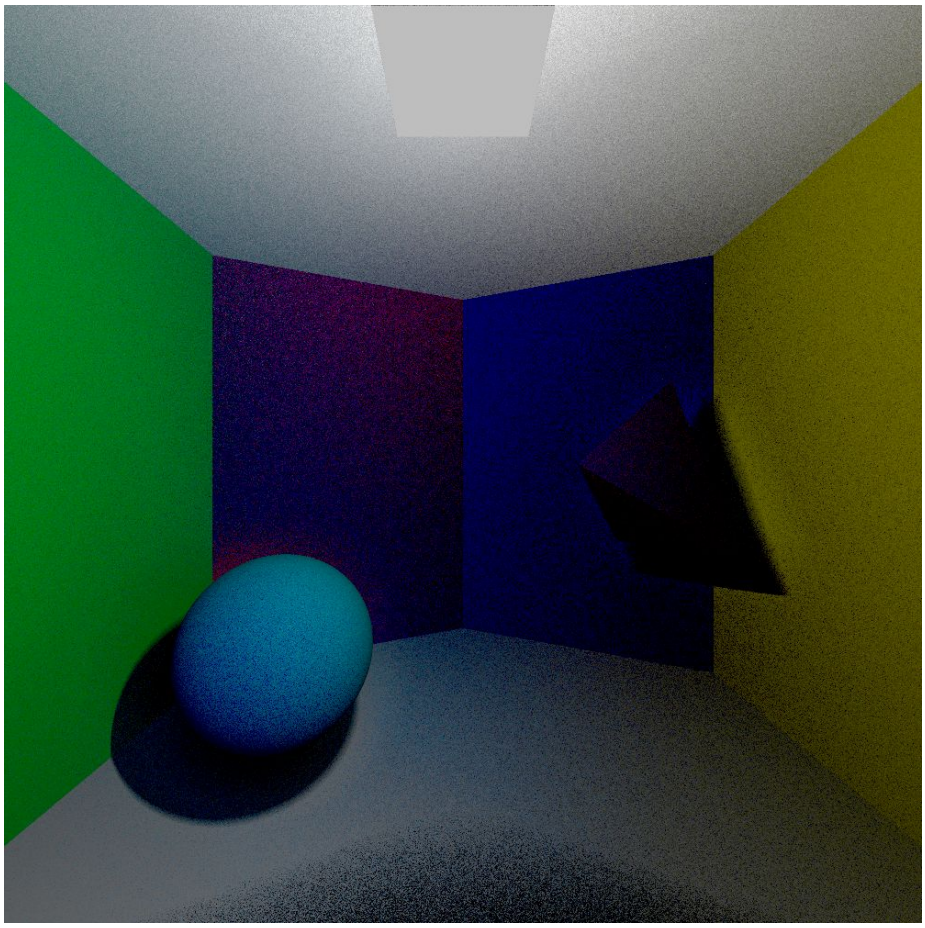

Figure 21: 8 reflected rays, 8 shadow rays and 8 rays per pixel

Rendering time for image
17: 577 sec (10 min)
18: 1245 sec (20 min)
19: 1217 sec (20 min)
20: 1379 sec (23 min)
21: 1632 sec (27 min)

4 Discussion

The implemented project only included Lambertian reflectors for surface integration. To achieve better global illumination, reflectors for specular, glossy and transparent surfaces is necessary to be included in the implementation as well. Specular surfaces reflect the light rays perfectly, which means the angle between the incoming light ray and the normal of the point on the surface, is equal to the outgoing angle between the normal and the reflected light ray. This phenomenon is also called the law of reflection. Specular surfaces reflect all the light further without any loss of light intensity or other information, and results in very sharp appearance of reflections on the surface. In addition to reflections, transparent materials introduce refraction models as well. Refraction is the change of direction of light when going from one material to another. Refraction introduces effects such as caustics, which has a huge impact of the photorealism in the rendered image. The angle of the refracted ray, depends on the density of the two adjacent materials, and follows Snell's law.

The larger number of rays used in the supersampling, the better quality of the image. But with every ray added to sample a pixel the computation time increases significantly since the added ray also needs to traverse through the scene the set number of iteration. However, the quality of the shadow can be improved by increasing the number of shadow rays without increasing the render time too much. This is because the shadow rays will not reflect after hitting the light.

The estimation of the indirect light contribution in the implementation is not a very good representation seeing there is only one sampling ray. A better result would have been achieved if the number of sampling rays would have been greater: When the number of sampling rays are more the variance of the estimation and the function is less. So the more sampling rays the more lifelike the images will be.

Advantages with Monte Carlo integration is that it handles a lot of visual effects, like soft shadows and color bleeding, which have a great impact of global illumination in rendered images. It is also possible to render any type of material with Monte Carlo Ray Tracing, such as diffuse, reflective, transparent and combinations of the three, making the final image look realistic. One disadvantage with the algorithm is that many rays are required to avoid aliasing and to get a accurate indirect lighting. Since the method uses a huge amount of sampled rays to achieve more better results, the method is time consuming.

References

[1] Lighthouse 3D. Ray-sphere intersection.

[2] Christopher De Angelis. Radiosity: A study in photorealistic rendering.

[3] Donald P. Greenberg Cindy M. Goral, Kenneth E. Torrance and Bennett Battaile. Modeling the interaction of light between diffuse surfaces.

[4] H. W. Jensen. Global illumination using photon maps.

[5] Scratchapixel. Global illumination and path tracing.

[6] Scratchapixel. An overview of the ray-tracing rendering technique.

[7] M. Cammarano H. Wann Jensen P. Hanrahan T. J. Purcell, C. Donner. Photon mapping on programmable graphics hardware.

[8] Stephen Merity Tegan Brennan and Taiyo Wilson. Monte carlo methods for improved rendering.