Global Illumination Rendering With Monte Carlo Ray-tracing

Per Blåwiik, Anneli Nilsson

Department of Science and Technology
Linköping University
2019-12-15

Keyword: Global Illumination, Monte Carlo Ray-tracing.

Abstract: This report describes the methods used to implement an advanced global illumination renderer using Monte Carlo ray-tracing and integration. The aim of the renderer is to provide a photorealistic image. The renderer uses direct, indirect and specular light simulation defined by the rendering equation, Whitted ray-tracing and Monte Carlo integration. The scene that was used to render images had implicit objects, spheres, and polygonal objects with Lambertian and Oren-Nayar material properties. There were also perfect mirror objects and one transparent, that uses the Fresnel equations. To determine when the light hits an object ray intersections are calculated, for triangles the Möller-Trumbore algorithm is used and the sphere intersection uses its geometry to determine intersections. Anti-aliasing is taken into consideration in this implementation, and is solved with supersampling and randomization. The result is a renderer based on global illumination techniques and unbiased schemes such as russian roulette and importance sampling. Given endless rendering time and samples, the rendered images converges to photorealism. Monte Carlo ray-tracing will always contain more or less noise based on the number of samples used.

1. Introduction

When recreating an image digitally one often strive for photorealism. Photorealism, in computer graphics, is a concept describing the phenomena when the image quality converges to an accurate and correct representation with increasing computer power, with the given specific methods. To achieve photorealism in a computer generated scene, one needs to implement a global illumination model. There are various techniques to simulate global illumination with different level of complexity. The implementation of global illumination is, however, a processing and memory expensive task which depends on how the implementation is done. The purpose of this report is to describe the implementation of a global illumination renderer using Monte Carlo ray-tracing and Whitted ray-tracing, in a scene containing different objects with various material properties.

1.1. Aim

The aim of this study is to use the mathematical equations and theories to implement a global illumination renderer with Monte Carlo ray-tracing and integration.

1.2. Global Illumination

The concept of global illumination consists of a number of components that are accounted for when illuminating a 3D-environment. These components are: direct illumination, indirect illumination and specular reflections. These types of illumination distribute flux, or light, across the entire scene in various ways from a light source. While a infinitesimal small area of flux is called radiance.

- **Direct illumination** is the flux that only bounce once in the scene after being emitted from the light source and then hits the view point.
- **Indirect illumination** is when flux from a light source bounce more than once in the scene and then hits the view point. In other words, indirect light is the refracted and reflected light.
- **Specular reflection** is light that has been perfectly reflected off a glossy surface.

Shadows are also a part of global illumination as well as caustics. Caustics are defined as concentrated flux in a specific area that has been refracted through a surface. However, caustics cannot be accurately represented by just Monte Carlo ray-tracing because it would produce a noisy result.

1.2.1. Implementation of Global Illumination

The major steps of implementing a global illumination are the following: define material properties and objects, light transportation and visual display. The first step is to generate the geometric environment of the scene, such as the room and any possible objects within it, and all the material properties of their surfaces. Examples of material properties can be transparency, color and reflectivity. The light sources are also defined in this step.

The second step is to calculate the light distribution in the scene by using the information about the light sources and surfaces. This step will result in radiometric values in the soon to be image plane, where radiometry is a kind of light energy. The third and the last step is to convert these radiometric values into pixel color.

2. Theoretical Foundation and Algorithms

In this section, all relevant techniques for the implementation of a Monte Carlo based global illumination renderer is explained.

2.1. Whitted Ray-tracing

The basic concept of ray-tracing is to follow a ray from the view point (camera position), through each pixel of the viewport image, into a 3D environment. If a ray intersects with a surface of an object, the radiance of the point is computed by casting a shadow ray towards all light sources (see Figure 1). If the surface material is specular, the ray is reflected and continues to bounce through the scene until it hits a diffuse surface or is terminated by a specified condition. The final pixel radiance is computed by summing up the radiance for each intersection point in the ray path. This recursive nature of including indirect light is known as Whitted ray-tracing [1]. More concretely, Whitted Ray-tracing recursively structures reflections and refractions from a ray into a tree structure, starting from each pixel.

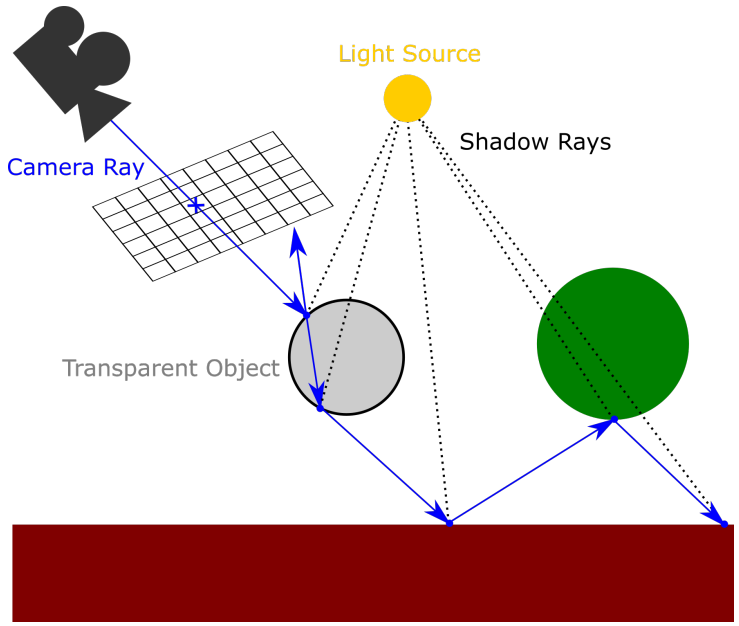


Figure 1: The path of the ray is visualized by the blue line and starts from the camera. For each surface intersection, a shadow ray is cast towards the light source. If the shadow ray intersects with another object before it hits the light source, the surface point lays in shadow.

2.2. Ray Intersection Test

When rays are projected into the scene, the rays will intersect with different objects. In this project two types of intersection are computed, the triangle and sphere intersection, because the spheres in the scene are implicit objects and the rest of the scene is made out of triangles.

2.2.1. Triangle Intersection

The majority of the scene contains triangles and to detect a ray intersection a formula called the Möller-Trumbore algorithm [2] is used. The definition of a ray $R(t)$ is seen in (1), O it the ray origin, normalized in the direction D and the vector t contains the intersection's distance and its (u, v) coordinates. Also, note that each triangle is described by three vertices V_0, V_1 and V_2.

$$R(t) = O + tD \tag{1}$$

The point $T(u, v)$ on a triangle is given by (2).

$$T(u,v) = (1-u-v)V_0 + uV_1 + vV_2 \tag{2}$$

The coordinates (u, v) are barycentric and must fulfill the conditions $u \geq 0$, $v \geq 0$ and $u + v \leq 1$. To find out if a ray is intersecting a triangle one needs to combine (1) and (2) that will result in (3).

$$[-D, V_1 - V_0, V_2 - V_0] \begin{bmatrix} t \\ u \\ v \end{bmatrix} = O - V_0 \tag{3}$$

The equation can be solved with Cramer's rule and will result in (4), where $E_1 = V_1 - V_0$, $E_2 = V_2 - V_0$, $T = O - V_0$, $P = D \times E_2$ and $Q = T \times E_1$.

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{PE_1} \begin{bmatrix} QE_2 \\ PT \\ QD \end{bmatrix} \tag{4}$$

2.2.2. Sphere Intersection

The surface detection when a ray collides with a sphere is calculate differently compared to a triangle, because a sphere is defined implicitly in the implementation. In other words a sphere only contains information about its center point and radius. The intersection detection is derived from (5), where $\mathbf{x}$ represents a sphere surface point, $\mathbf{c}$ is the sphere center and r is the sphere radius.

$$||\mathbf{x} - \mathbf{c}||^2 = r^2 \tag{5}$$

The term $\mathbf{x}$ is given by (6) where $\mathbf{o}$ is the starting point of a ray together with the normalized direction $\mathbf{l}$. The factor d is a point $\mathbf{x}$ on the ray.

$$\mathbf{x} = \mathbf{o} + d\mathbf{l} \tag{6}$$

d can be calculated by using (6) in (5), which will result in (7).

$$d = -\frac{2\mathbf{l}(\mathbf{o} - \mathbf{c})}{2} \pm \sqrt{\left(\frac{2\mathbf{l}(\mathbf{o} - \mathbf{c})}{2}\right)^2 - ((\mathbf{o} - \mathbf{c})^2 - r^2)} \tag{7}$$

2.3. Rendering Equation

The general idea of a global illumination renderer is derived from the formula shown in (8) [3].

$$L(x \leftarrow \omega) = L_e(x \leftarrow \omega) + \int_{\omega 1} f^*(x_1, -\omega, \omega_1) L(x_1 \leftarrow \omega_1) d\omega_1 \tag{8}$$

The term $L(x \leftarrow \omega)$, or $L(x_1 \leftarrow \omega_1)$, is the radiance that arrives at a point x (x_1) from the direction ω (ω_1), while L_e denotes the radiance emitted by the light source that arrives at x from the direction ω. $f^*(x_1, -\omega, \omega_1)$ is the fraction of reflected radiance from the surface point x_1 from a direction ω_1 into the direction $-\omega$. f^* also depends on two factors, the material property (f_r) and a geometric factor (9). The inclination angle θ_1 is the angle between the normal at x_1 and ω_1.

$$f^*(x_1, -\omega, \omega_1) = f_r(x_1, -\omega, \omega_1) cos\theta_1 \tag{9}$$

The rendering equation is, however, defined as an integral and it is not possible to make a rendered image with just this. The solution is to use a numerical interpretation of the formula, like Monte Carlo ray-tracing.

2.4. Fresnel Equation

There are specific equations called *Fresnel equations* that calculate the ratio between refracted and reflected light, this applies only to smooth transparent objects. The Fresnel equations utilize Snell's equation for refraction where θ_1 is the incident angle of the incoming ray. n_1 and n_2 are the respective media's refractive index. Equations (10) and (11) describe the perpendicular and parallel components of polarized light [3].

$$R_s = \left(\frac{n_1 cos\theta_1 - n_2 \sqrt{1 - ([n_1/n_2] sin\theta_1)^2}}{n_1 cos\theta_1 + n_2 \sqrt{1 - ([n_1/n_2] sin\theta_1)^2}} \right)^2 \tag{10}$$

$$R_p = \left(\frac{n_1 \sqrt{1 - ([n_1/n_2] sin\theta_1)^2} - n_2 cos\theta_1}{n_1 \sqrt{1 - ([n_1/n_2] sin\theta_1)^2} + n_2 cos\theta_1} \right)^2 \tag{11}$$

The coefficient of total reflection (12) is a combination of R_s and R_p, while $T = 1 - R$ denotes the transmitted amount of radiance.

$$R = \frac{R_s + R_p}{2} \tag{12}$$

2.5. Bidirectional Reflectance Distribution Function

There are a couple different types of surfaces in the scene that were used that were defined in different ways. The diffuse surfaces in the scene were either defined as a Lambertian or Oren-Nayar surfaces. Other surfaces are perfectly reflective (mirrors) or transparent. The case when a surface is a perfect mirror, implies that the radiance is left unchanged when it hits the surface and then bounces away. Transparent surfaces reflect part of the incoming ray and let the other part pass, or refract, through the surface.

Lambertian and Oren-Nayar models are examples of *Bidirectional Reflectance Distribution Function*, even referred as BRDF. Lambertian surfaces have a uniform distribution of radiance giving a smooth result, while Oren-Nayar simulate a rough surface, see Figure 2.

Figure 2: Comparisons between a real image, Lambertian and Oren-Nayar surfaces. [4]

In the Lambertian model (13) the direction of the incoming and outgoing rays have no influence on the light distribution on the surface. The BRDF in this case only have a constant reflection coefficient ρ in the interval $[0, 1]$, which is normalized with π.

$$f_r = \frac{\rho}{\pi} \tag{13}$$

Oren-Nayar reflectors can simulate rough surfaces more accurately than the Lambertian model, because the surface is structured with V-shaped micro-facets that results in a visualization of a rough surface [4]. The model has Guassian distribution of the microfacets' angle orientation and is defined in (14). $\phi_{in}, \theta_{in}, \phi_{out}$ and θ_{out} are the directions of the incoming and outgoing rays, while α and β are defined as $\alpha = max(\theta_{in}, \theta_{out})$ and $\beta = min(\theta_{in}, \theta_{out})$.

$$f_r(x, \omega_{in}, \omega_{out}) = \frac{\rho}{\pi}(A + Bmax(0, cos(\phi_{in} - \phi_{out}))sin\alpha sin\beta) \tag{14}$$

The constants A and B are calculated with (15), where σ represents the standard deviation of the Gaussian.

$$A = 1 - \frac{\sigma^2}{2(\sigma^2 + 0.33)}, \quad B = \frac{0.45\sigma^2}{\sigma^2 + 0.09} \tag{15}$$

2.6. Monte Carlo Integration

The implemented renderer presented in this paper is predominantly based on the *Monte Carlo integration* technique *Importance Sampling*. Monte Carlo integration applies a non-deterministic approach, meaning that the integrated outcome is an approximation of the correct value reached by picking random points based on the definition of the integrated function. By picking an infinite amount of random points (samples), the integration will converge to the correct value. This is the main motivation for using the method: the random points result in an unbiased final value which contributes to photo-realism.

In the case of applying the Monte Carlo method with the Whitted ray-tracing scheme described in section 2.1, each ray-intersection on a diffuse surface casts N number of sample rays to collect incoming radiance (or outgoing importance), see Figure 3.

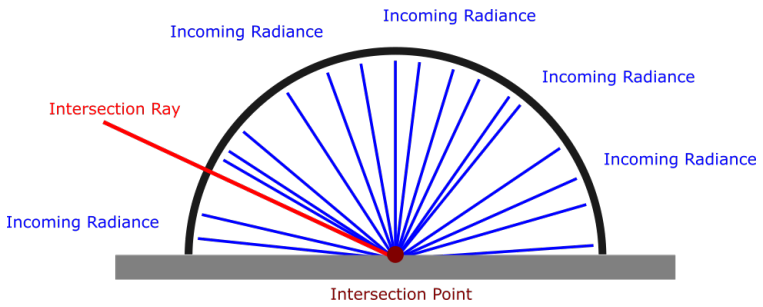


Figure 3: An illustration of the incoming radiance/outgoing importance for all diffuse intersection points. The rays are randomly sampled over the entire hemisphere.

The estimator for the Monte Carlo sampling is given by equation (16):

$$< E_P >= \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{p(X_i)}, \tag{16}$$

where $f(X_i)$ is the function to integrate (total amount of radiance on intersection point P), $p(X_i)$ is the *probability distribution function* (PDF) for the hemisphere, N is the number of random samples and X_i is the random variable (e.g. random point on the hemisphere).

In this case, the PDF p is constant since all sample rays have the same probability of being generated. Since the solid angle of a hemisphere is given by 2π, an integration of the constant p over the hemispheres give the solution:

$$\int_{\omega}^{2\pi} p d\omega = p(2\pi - 0) = 1 \Leftrightarrow p = \frac{1}{2\pi} \tag{17}$$

2.6.1. Importance Sampling

The main idea of importance sampling is to generate more samples where the integrated function is important, which leads to lower variance. This translates to: generate samples where the integrated function is similar to the PDF (important samples). This technique greatly reduces the noise compared to uniformly distributed sampling. In practise it means that the samples are generated based on the PDF.

Start by expressing the solid angle for the hemisphere in polar coordinates: $d\omega = \sin\theta d\theta d\phi$, where θ is the inclination angle, ϕ is the azimuth angle and $d\omega = 2\pi$ according to equation (17). The PDFs for θ and ϕ are then given by:

$$p(\theta) = sin\theta, \quad p(\phi) = \frac{1}{2\pi}$$

By inverting the *cumulative distribution functions* (CDF) of the above PDFs the randomly generated inclination angles θ and azimuth angles ϕ are given by equation (18) and (19):

$$\theta = \arccos(1 - r), \tag{18}$$

$$\phi = r * 2\pi, \tag{19}$$

where $r \in [0, 1]$ is a uniformly distributed random number. The final step is to simply transform the generated direction to the world coordinate system.

2.6.2. Russian Roulette

To terminate the recursion of the Monte Carlo sampled rays, an unbiased method is required for maintaining the photorealism. This is solved by a

technique called *Russian Roulette*, where the rays are randomly terminated based on an absorption probability α of a surface point. First, generate a random number $r \in [0, 1]$, then, if $r > 1 - \alpha$ the ray is terminated. To account for the reduced importance, the BRDF function ρ is divided by the probability of a ray not being terminated.

2.7. Anti-Aliasing and Supersampling

The mathematical descriptions of the geometries in the scene are continues. If a limited amount of rays are cast through a limited amount of pixels to describe a continues line, the effect of aliasing will occur (depending on the level of detail). If only one ray is emitted per pixel, the aliasing is primarily reduced by increasing the resolution (adding more pixels). This is only possible to an extent and therefore a technique called *supersampling* is used to divide each pixel into smaller sub-pixels (simulating higher resolution). The average value of the sub-pixels are then used to describe the real pixel, see Figure 4.

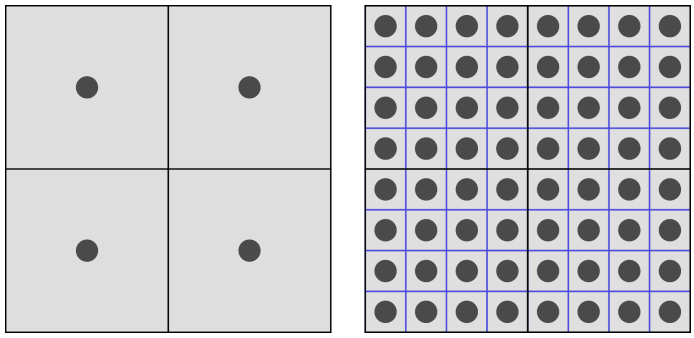


Figure 4: To the left is four ordinary pixels with one ray (black dot) through each pixel. To the right, each of the four pixels on the left has been subdivided into 16 sub-pixels. This technique simulates higher resolution by averaging the sub-pixels into one.

Another available method is to smooth out sharp edges by emitting each of the rays in different pseudo-random directions. This method is called ray randomization and is illustrated in Figure 5. Just as for supersampling, all random samples are being averaged to represent the real pixel.

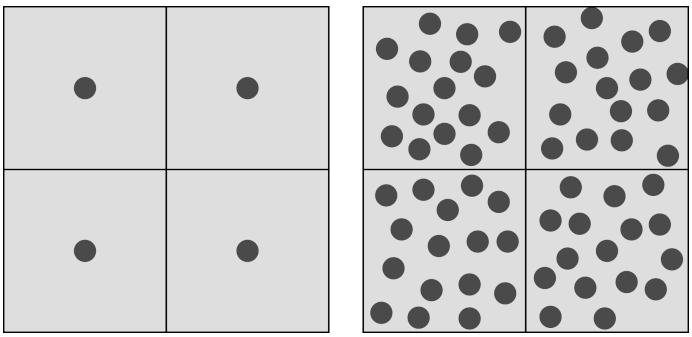


Figure 5: To the left is four ordinary pixels with one ray (black dot) through each pixel. To the right, each of the four pixels has been sampled into 16 random ray directions.

Naturally the best result is yielded by combining the two methods since the supersampling is still becoming aliased for certain cases and the random rays could end up being distributed unevenly.

3. Result and Discussion

The implemented Monte Carlo ray-tracing renderer is based on the theory presented in previous sections of the paper and will converge to a photorealistic image if using unlimited rendering time (infinite number of samples). In this section, certain visual effects on rendered images will be presented and discussed.

3.1. Rendering Result

The final scene (see Figure 6) is rendered in the resolution 1024x768 pixels, using 512 samples per pixel with randomized directions within each pixel. The only light source in the scene is a white rectangle area light. Maximum number of ray bounces were set to 7 to prevent never ending mirror loops. The spheres are implicit objects and the rest are polygonal objects. The walls, ceiling and floor are using diffuse Lambertian surfaces and the white sphere has a diffuse Oren-Nayar surface. There are two perfectly reflective surfaces: a tetrahedron and a sphere. The last sphere has a fully transparent surface similar to the material of glass. The renderer utilizes OpenMP[5] for parallel computations to decrease the rendering time.

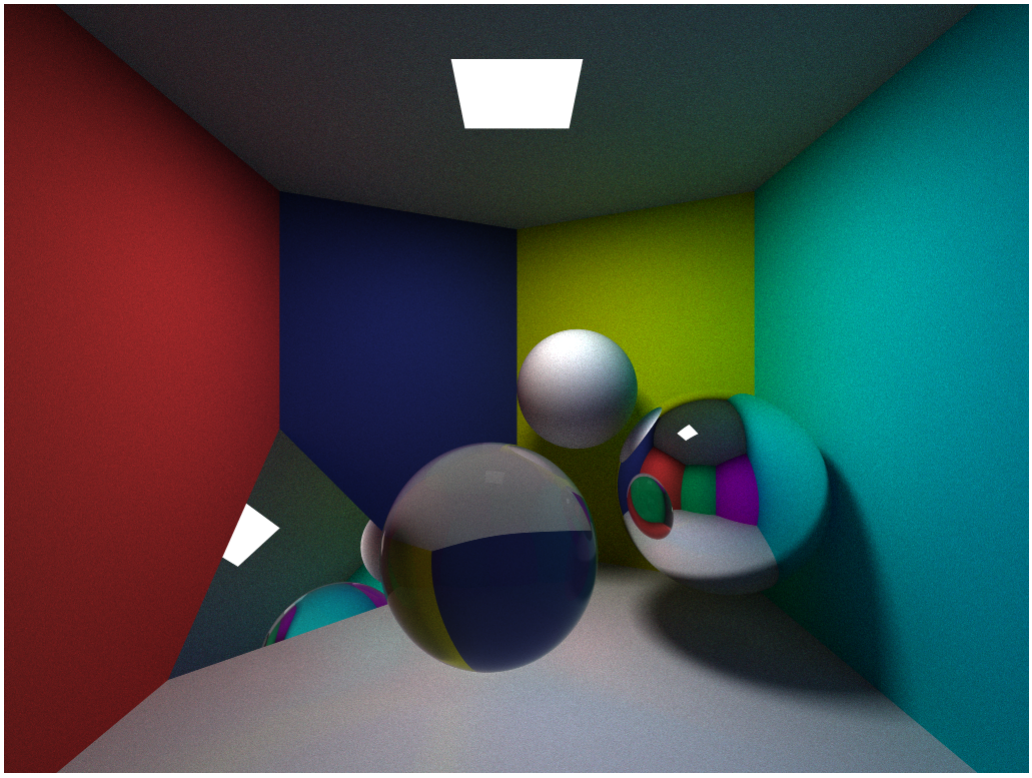
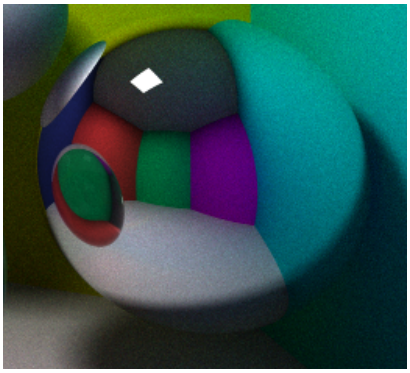


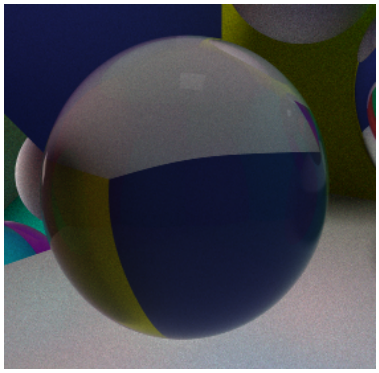
Figure 6: Scene rendered with Monte Carlo ray-tracing using 512 samples per pixel and Russian roulette for terminating rays. Rendering time: 143 min.

3.1.1. Reflection and Refraction

A zoomed in view of the sphere with a reflective surface from Figure 6, can be seen in Figure 7a. The surface is perfectly reflective and shows parts of the room that is not visible from the original view. We can also see the refractive effect of the fully transparent sphere, through the same reflection. This is an outcome of the recursion used in the path tracing. Light refraction in the transparent sphere is more clearly seen in Figure 7b. We can even see the shadow of the white sphere on the yellow wall through refracted light. As explained in section 2.4, a part of the light on a transparent object is reflected. This effect creates a vague reflection of the light source in the ceiling as well as the white sphere, as seen in Figure 7b.



(a) Perfect reflective material.



(b) Transparent material.

Figure 7: Objects with perfect reflective material and transparent material.

3.1.2. Color Bleeding

The white sphere with the Oren-Nayar BRDF in Figure 8 shows the effect of color bleeding. This is possible since the path tracing is recursive and each recursion uses Monte Carlo sampling to collect incoming radiance. If a white surface point is close to a e.g. red object, a large amount of the incoming rays will contribute to the color red (indirect illumination).

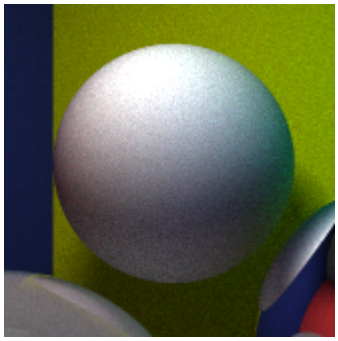


Figure 8: Sphere with a diffuse material showing the effects of color bleeding in the scene.

3.2. Noise

The implementation of the Monte Carlo algorithm in this project produce noise in the render result. The amount of noise depends on how many samples are used to render the image, as seen i Figure 9 with 64 samples which is

eight times less than the final rendering in Figure 6. As explained in section 2.6, in a real application where the number of samples are limited, the result will never be entirely noise free. However, with a large enough amount of samples, the noise will barely be noticeable. Most importantly, in the end the noise hides artifacts and is unbiased which contributes to maintaining photorealism.

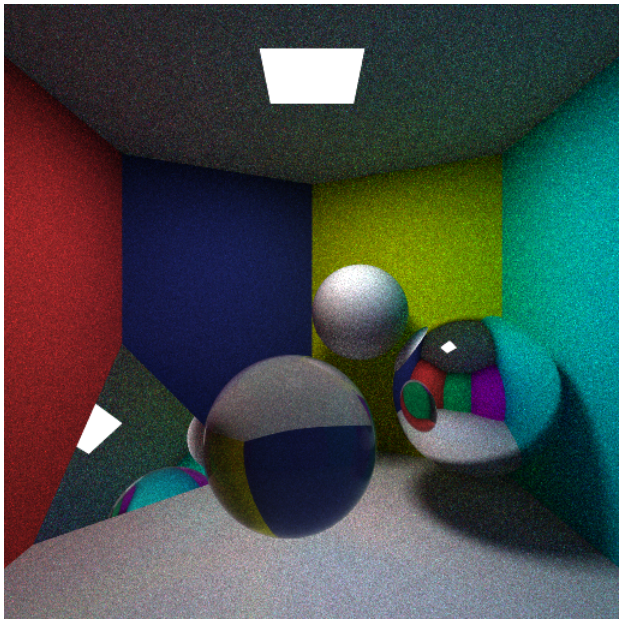


Figure 9: Render of the scene with 64 samples and resolution 512x512 pixels.

3.3. Further Implementation

The implementation of the global illumination renderer can be improved by adding methods to calculate caustics in the scene, which would improve the result of transparent objects when light travels through them. The implementation of caustics can be done with photon mapping, which needs a kd-tree structure to store the photons emitted into the scene. Photon mapping would also improve the rendering time, because it reduces the number of shadow rays and therefore less calculation needed to be done.

4. Conclusion

The projects purpose was to implement a global illumination renderer and render a scene with an area light source, implicit and polygonal objects with Lambertian and Oren-Nayar surface BRDFs, perfect mirrors and a transparent object. The renderer uses relevant equations to create an adequate photorealistic image. The Monte Carlo technique renders an image that will always contain noise, however humans have a tendency to ignore it if the amount is relatively small.

References

[1] Whitted JT. An Improved Illumination Model for Shaded Display. Graphics and Image Processing. 1979;.

[2] Möller T, Trumbore B. Fast, Minimum Storage Ray-Triangle Intersection. SIGGRAPH. 2005;.

[3] Dutré P, Bala K, Bekaert P. Advanced global illumination, second edition; 2006.

[4] Oren M, Nayar SK. Generalization of Lambert's Reflectance Model. SIGGRAPH. 1994;.

[5] OpenMP - The OpenMP API specification for parallel programming;Available from: https://www.openmp.org/.