Global illumination and rendering

TNCG15

Ronja Grosz
rongr946
Isabell Jansson
isaja187

April 12, 2016

Abstract

This paper discusses several global illumination techniques used in computer graphics, such as Radiosity, Whitted ray tracing, Monte Carlo ray tracing and Photon mapping with a two-pass rendering method. A more detailed description of Monte Carlo ray tracing is described along with Monte Carlo integration, Russian roulette and the use of importance and shadow rays.

The paper also discusses an implementation of a Monte Carlo ray tracer. Different aspects of the ray tracing where evaluated such as the amount of shadow rays, the number of importance rays launched through a pixel and the image resolution.

The implementation resulted in a Cornell box containing boxes, spheres and a quadratic area light source. The more shadow rays that are launched from the light source, and the more rays that a launched per pixel, the better result. The image resolution is also critical for a good result.

The result can be further improved by implementing a Whitted ray tracer in order to get perfect reflections and refractions. In order to get an improved representation of the objects, an Oren-Nayar reflectance model should be implemented. For more details in the scene, the amount of rays per pixel and the amount of shadow rays can be increased. This will also increase the run time drastically and the quality of the result is therefore always a compromise.

1 Introduction

There are several methods to use when rendering a scene. An introduction to global lighting models follows below.

1.1 Global lighting models

1.1.1 Radiosity

Radiosity is used to render scenes with Lambertian reflectors. The method simplifies the rendering equation to an equation using radiosity. This is done by using the definition of irradiance and assuming that the surface is a Lambertian reflector. A Lambertian reflector emits radiance in all directions which makes the directional dependence of the radiance irrelevant. The definition of radiosity is then used to find the radiosity equation for Lambertian reflectors.

To solve the radiosity equation, all other points in the scene have to be taken to account. To simplify the computations the continuous surfaces are approximated by flat triangles. Each triangle has a surface area, a normal direction and a reference point. The radiosity is assumed to be constant on each triangle which will result in flat shading.

To evaluate the radiosity, the interaction between all other visible triangles have to be tested together with the form factor which expresses how easily the light is transferred between triangles. Every pass with the radiosity equation introduces one more reflection.

1.1.2 Whitted ray tracing

The method considers perfect reflections and perfect refractions. In Whitted ray tracing importance rays are launched from the camera, through a pixel, into the scene. When the importance ray hits a surface, it will either be reflected, refracted or both, depending on the object's material. The outgoing importance ray has the same angle as the incoming importance ray, since it is a perfect reflector. The angle for the refracted ray is calculated by Snell's law. All ray intersections are stored in a tree structure where the root correspond to the original importance ray launched from the eye and the nodes correspond to all surface intersections. When a ray hits a light source, hits an object that is not a perfect reflector, the importance decreases below a certain threshold or when a maximum number of iterations have been reached, the ray tracing stops.

The radiance in the intersections points are evaluated by following the ray path backwards, starting at the leaf nodes in the tree. In each branch of the tree the direct, reflected and refracted light are combined to evaluate the radiance. To test if an intersection point should have, a direct light contribution shadow rays are used.

1.1.3 Monte Carlo ray tracing

Monte Carlo ray tracing is a fast method that works for all BRDFs in contrast to Whitted ray tracing. The main difference from Whitted ray tracing is how the outgoing importance ray is calculated. In Whitted ray tracing, the angle for incoming importance and outgoing importance is identical. In this method, the angle for the outgoing importance is calculated through two random numbers. This method will be discussed in more detail further on.

1.1.4 Photon mapping with a two-pass rendering method

With pure Monte Carlo ray tracing, reflected rays are shot into random directions. This may lead to that caustics could be missed when rendering the scene. To avoid this problem an alternative is to use photon maps with a two-pass rendering method. The aim is to store every hit in a three dimensional map, often a kd-tree, and then use the map to render the scene. This is done in two passes and can also be used to render scenes with participating media [3].

In the first pass the photon maps are constructed by shooting radiance from the light sources into the scene. The rays can interact with participating media, or be reflected or transmitted by objects in the scenes. If the ray interacts with participating media, Russian roulette decides whether the ray is scattered or absorbed. When a ray interacts with the scene, the incoming flux and direction are stored in the maps [3].

In the second pass the photon maps are used to calculate the radiance in every hit position in the scene. The caustics are rendered directly. The stored flux is used to estimate the photon density around a point in the scene which corresponds to the reflected radiance [3].

1.2 Monte Carlo ray tracing

1.2.1 Monte Carlo integration

In order to solve the rendering equation, Equation (1), the integral has to be solved numerically. This can be done with Monte Carlo integration which estimates the integral using a probability distribution.

From the rendering equation the probability distribution function is chosen according to Equation (2). This part of the integral is the only part that can be used since the BRDF may not result in an invertible cumulative distribution function and L is initially unknown. The cumulative distribution function is found by integrating Equation (2). The result can be seen in Equation (3), and is used to map two random numbers $u, v \in [0, 1[$, uniformly distributed, to two angles $\theta \in [0, \frac{\pi}{2}], \varphi \in [0, 2\pi]$, Equation (4).

$$L(x \to \omega_{out}) = L_e(x \to \omega_{out}) + \int_{\Omega} f_r(x, \omega_{in}, \omega_{out}) L(x \leftarrow \omega_{out}) \cos\theta_{in} d\omega_{in} \tag{1}$$

$$p(\theta, \varphi) = \frac{1}{\pi} \cos\theta \tag{2}$$

$$p(\theta, \varphi) = \frac{\varphi}{2\pi}(1 - \cos^2\theta) \tag{3}$$

$$\varphi = 2\pi u \qquad \theta = \cos^{-1}(\sqrt{v}) \tag{4}$$

1.2.2 Russian roulette

Russian roulette is an unbiased method used to terminate the ray path. This is done by rescaling the interval for the random numbers $u, v \in [0, 1[$ to $u, v \in [0, p[$, where $p \in [0, 1[$. This will decrease the estimator and the error has to be compensated by multiplying the outgoing importance by the factor $\frac{1}{p}$. The estimator is then used to determine if the ray is terminated or not.

1.2.3 Ray tracing

To evaluate the radiance in a point on a surface, two components have to be taken to account, direct light and reflected light. The direct light corresponds to the L_e-term in the rendering equation and the reflected light corresponds to the integral part in the equation.

1.2.3.1 First step: Trace importance rays through the scene Importance rays are launched from the camera, through a pixel in the image plane, into the scene. In order to reduce aliasing, a random noise effect was added to the ray direction. By shooting n rays through every pixel with a small random noise, and divide the summarized result by n, the aliasing was reduced. When the ray hits an object, Russian roulette decides whether the ray should be terminated or continued. If the ray continues, the ray will be reflected (transmitted rays are not yet considered) into a new direction.

To obtain the new direction for the reflected importance ray Monte Carlo integration is used. The random angles, found by Monte Carlo integration, represent a random direction for the incoming radiance. The new random direction is found by rotating a tangent vector to the hit position with random angles theta and phi. Since the radiance is parallel to the importance and they obey the same physical laws, this direction is the direction for the outgoing importance as well. The importance for the outgoing ray is calculated according to Equation (5), p corresponds to the compensation for the Russian roulette.

$$W_2 = \frac{\pi}{p} f_r W_1 \tag{5}$$

1.2.3.2 Second step: Launch shadow rays from the light sources When all ray paths have been terminated, shadow rays are launched from the importance intersection points to the light sources. The algorithm goes backwards, starting with the last ray in the ray path. The last ray will not have a reflected light component. If the intersection point is visible from the light source, the light source will contribute to the L_e-term, i.e. it will have a direct light contribution from the light source. To calculate the reflected light in the previous intersection point Equation (6) is used. The algorithm goes backwards until the radiance for the original ray is calculated. In every ray intersection point, the radiance is a sum of the contribution from the reflected ray and the direct light.

$$L = \frac{L_{prev} W_{prev}}{W} \tag{6}$$

2 Background

2.1 Scene

The scene consists of three different views. The world, model and camera view. There are three separate views to make it easier to place objects, light sources and the image plane in different sizes and at different positions.

The camera and the image plane are created in the view coordinates and are transformed into world coordinates. In the world coordinates, the ray's starting position and direction is calculated by using the camera and the image planes coordinates in the world coordinate system. The starting position for the rays is the camera position and the direction is calculated from the camera position and a position in the image plane.

It is in the model view where all ray intersection computations take place. To test the intersection between the objects and lights in the scene and the ray, the ray is transformed from the world view to each of the objects' or lights' model view. The intersection is then calculated between the ray and all objects and lights in the scene.

Every object and light has their own model view. It is in the model view that the object or light is created. The objects are built in the origin of the model view.

2.2 Objects

There are two sets of objects in this project. There are spherical objects and objects built from triangles. These two types of objects have different intersection algorithms.

A triangle-based object is described by two classes. One class that describes the structure of the triangles, for example a box, and its characteristics and one class for the triangles. The triangle class has three vertices, a normal and a color that the triangle is based on. It also has an intersection function that is based on the Möller Trumbore intersection algorithm between a line and a triangle [1]. The triangle class is used for every triangle based object and light source. The object also has a class for describing the object using triangles. The class has a vector of triangles that describe the object and a function for intersection with these triangles. It also contains the object's BRDF.

A spherical object is not triangle-based, instead the surface is parametric. Therefore the sphere class does not depend on the triangle class. The sphere class has a radius, BRDF and a color that described the sphere and an intersection function based on a line-sphere intersection [2].

2.3 Rays

The ray class contains a starting position, direction and the hit position to describe the ray. The ray also has a pointer to its parent ray and children rays, which are the transmitted and the reflected ray (transmitted ray is not yet used in the code). So that the ray path can be saved and visited.

For every pixel in the image plane, a ray is shot through it. The ray is tested against every object and light source in the scene whether or not it intersects with that object. If the ray intersects with several objects in the scene, the one that intersects with the ray first has its hit point saved in the ray, this corresponds to the shortest possible ray.

2.3.1 Importance rays

If the ray hits a light source, the ray will not continue. If it hits an object, the chance of the ray continuing is decided through Russian roulette. If the ray continues, the random angles are calculated using the Monte Carlo integration scheme. The angles are then used to rotate the surface normal to point in the new direction. A new ray is created with the new direction and is set as the reflected child ray for the previous ray. The importance transferred to the new ray is calculated through Equation (6).

2.3.2 Shadow rays

When the last ray in a ray path hits a light source, the direct contribution to the ray is from the light source. If the last ray is terminated on an object, shadow rays are sent from the object to light sources in the scene. If the path is not blocked by another object, the contribution from the light source is calculated by $f_r \cdot L_e$.

The ray path is then traced backwards recursively. How much of the contributed light L from the child ray that is transferred to the parent ray is calculated using the relationship between importance and radiance. Shadow rays are also sent to each of the objects that have been hit on the ray path. The total light contributed is then summed when reaching the first ray in the ray path.

2.4 Ray intersection

The ray intersection is calculated differently depending on which object it intersects with. For triangle objects like boxes, the Cornell box and light sources Möller Trumbore intersection has been used. For spherical objects a simpler solution with line-sphere intersection has been used. The following subsections will go through this in more detail.

2.4.1 Triangular objects

An intersection between a ray and a triangular object is made through the Möller Trumbore intersection algorithm. When calculating the intersection between a ray and for example a box, the intersection algorithm is run for every triangle in the object. The triangle that is the closest intersection with the ray is the only triangle that counts as intersected with. The Möller Trumbore algorithm compares the ray and the triangle. If they are parallel, or the triangle is back-facing there is no intersection. It then tests if the ray intersection lies outside of the triangles bounds, if it does not there is an intersection between the ray and the triangle.

2.4.2 Spherical objects

The intersection between a spherical object and a ray is calculated through the line-sphere intersection algorithm. The ray is transformed to the object world. The line-sphere equation is made through inserting the line equation into the sphere equation, see Equation (7).

$$|\overline{r} - \overline{r_{sphere}}|^2 = R^2 \tag{7}$$

To see if the ray intersects with the sphere c is calculated through Equation 8, where d is the ray direction, o is the ray starting point, r is the radius and o_s is the spheres center point. If c is zero there is only one solution and if c is above zero there are two solutions, where the closest intersection point is chosen. If c is negative, the ray does not intersect with the sphere.

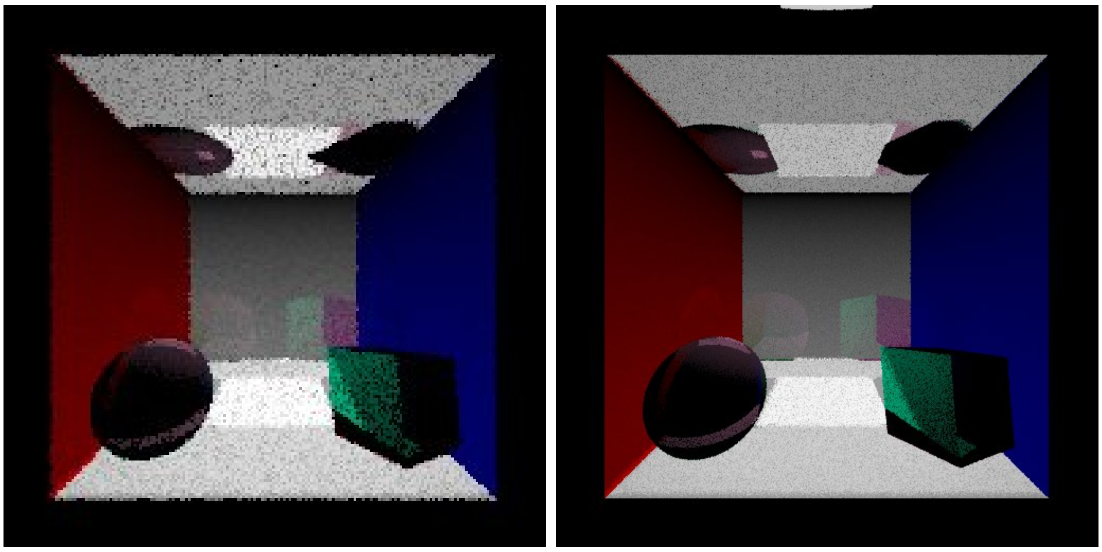
$$c = (d \cdot (o - o_s))^2 - d^2 \cdot (o^2 - r^2) \tag{8}$$

2.5 Image

The light calculated in every pixel is saved in a vector. The light is then normalized by dividing with the largest L term. To improve the result, the intensity is scaled so that the output pixel's intensity corresponds to the square root of the calculated pixel intensity. The information in the color vector is then written into a BMP image.

3 Results and benchmarks

The result when varying the image resolution can be seen in Figure 1a and 1b. The resolution only affects the the quality of the image, not the colors. The rendering time for image 1b was 450 seconds and for image 1a it took 56 seconds.

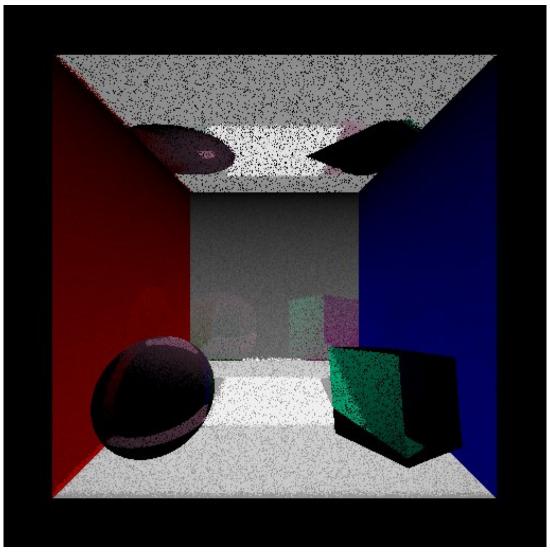


(a) A sphere and a box.
Image size:200x200,
Rays per pixel: 3,
Number of shadow rays: 3.

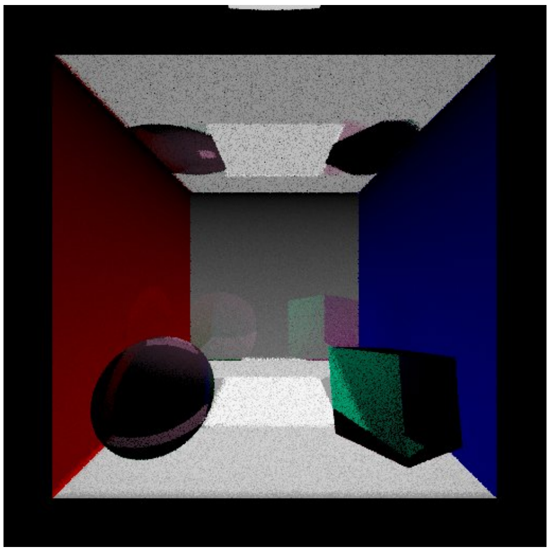
(b) A sphere and a box.
Image size:512x512,
Rays per pixel: 3,
Number of shadow rays: 3.

Figure 1: The scene rendered with different resolutions

Before the ray direction was improved with a small noise, the result contained more noise, Figure 2a. The improvement can be seen in Figure 2b.



(a) A sphere and a box.
Image size:512x512,
Rays per pixel: 1 (No ray randomization),
Number of shadow rays: 3.



(b) A sphere and a box.
Image size:512x512,
Rays per pixel: 3,
Number of shadow rays: 3.

Figure 2: The scene rendered with different amounts of rays per pixel

4 Discussion

The project could be further developed by implementing whitted ray tracing to get perfect reflections and refractions. This would allow the scene to look closer to reality.

The Oren-Nayar reflectance model could be implemented to give a better representation of the objects. Oren-Nayar produces a more realistic look where objects appear more matte than objects rendered with the Lambertian model.

Photon mapping could be implemented to give the scene a simulation of realistic

The result could improve by rendering the image with even more rays per pixel position. This would remove the noise in the image. As of now at most three rays per pixel has been used where the starting position has been randomized so to not fire the ray three times from the same starting position. A better resolution for the image could also achieve more details in the scene. As for now a resolution of at most $512 \cdot 512$ pixels has been used.

References

[1] Möller–Trumbore intersection algorithm
https://en.wikipedia.org/wiki/M%C3%B6ller%E2%80%93Trumbore_intersection_algorithm

[2] Line-sphere intersection
https://en.wikipedia.org/wiki/Line%E2%80%93sphere_intersection

[3] Efficient Simulation of Light Transport in Scenes with Participating Media using Photon Maps
Henrik Wann Jensen and Per H. Christensen