Implementing a Witted Ray Tracer

Mirza Talovic
mirta980@student.liu.se

Anders Nord
andno922@student.liu.se

TNCG15 - Advanced Global Illumination and Rendering
Linköpings Universitet

January 13, 2013

Abstract

This report covers the implementation of a Whitted ray tracer made in the course Advanced Global Illumination and Rendering at Linköpings University. The task was to implement a ray tracer that could handle global illumination. We chose to implement a Whitted ray tracer with perfect reflection and refraction.

Contents

Structure of this report

The first section of this report will introduce the subject of digital images by briefly explaining different rendering methods.A detailed description regarding the implementation of a Whitted ray tracer is presented in section two. After that, some benchmarks are presented.And finally section four,covers the conclusions drawn.

1 Introduction

Since the beginning of computer graphics there have been an desire to develop methods that produce photo realistic images. With todays knowledge, algorithms and computational power, this goal has been reached. This paper will present how a Whitted ray tracer has been implemented. In order to gain deeper understanding of the underlying algorithms the fundamentals of this technique will be discussed. Ray tracing lies in category referred to as three-dimensional computer graphics. In summary, it is the generating of images from a 3D scene, that can contain complex objects, different types of materials and several light sources. Transport and scattering are the two main components required for producing photorealistic images. Transport - describing the light propagating in the scene. Scattering - to describe how the transported light should interract with different materials and objects in the scene.

1.1 Different rendering methods

Photorealism is very much dependent on what type of lightning method that is used. A global lightning model gives a more realistic result than a local model, since it can better represent how light propagates and interacts in reality. The goal is to achieve a representation of light that is physically correct, which is not that easy in computer graphics. There are several different methods capable of computing scenes with this type of light-representation: Whitted ray tracing, Monte Carlo ray tracing, Two pass rendering and photon mapping are three examples of such. These are all different solutions to the *rendering equation* seen in 2.4.2. All of the methods mentioned are breifly explained below.

1.1.1 Local lightning models

Local lightning models only consider interaction between lights and objects, which means that light arriving at a point can only come directly from a light source. Local lightning models usually consists of the following three components,an ambient, a specular and a diffuse term.

The ambient term refers to light that has been scattered equally in all directions at a surface. In general this means that ambient light has bounced

off many surfaces before it reaches the observer.

Diffuse light on the other hand, is light that generally comes from one direction. Once it hits a surface it gets scattered equally in all directions. The results appears equally bright, regardless of the observer location.

When specular light hits a surface it tends to reflect in a particular direction. It appears as a shiny spot on the material it interracts with, since the light gets concentrated at a small patch of the object. Different materials have different specular properties. For example, metal and plastic have a high specular component while chalk for instance almost have none.

The simplest shading methods are the ones calculating the intensity based on indicent angle of light. This is achieved by only using a surface color for each polygon and a point light source [5]. Two methods in this category are Gouraud and Phong shading. Gouraud shading interpolates the color by averaging the vertices of the polygon. Each vertex have assigned colors, and these are blended across the surface of the polygon. The result is far more smooth compared with flat shading, since each vertext typically have minimum three neighboring polygons. Phong shading on the other hand, uses the colors of neighboring pixels to avarage the current pixel. When light gets reflected in the Phong model, it consists of a diffuse, a specular and an ambient component. By calculating the linear combination of the three terms, the intensity of the current point can be obtained.

The Phong model is an improvement when compared to Gouraud shading, since it is capable of calculating a better approximation of the shading for a smooth surface. The advantage of Gouraud shading is that it is computationally less expencive. Local lightning models like these are relatively computationally cheap, but does not account for indirect lighning. To achieve this a global lighning model is needed.

1.1.2 Global lightning models

Global lightning models can unlike local lightning models also handle indirect light. This refers to light coming not only directly from a light source, but also from surrounding objects. By using this type of lightning, based on the full scene, it is possible to simulate effects such as color bleeding, soft shadows and caustics. Since these effects add realism to rendered images, global illumination methods are a requirement to reach photorealism. Many

global illumination models that are used today is based on a ray tracing algorithm introduced by Turner Whitted, in 1979. Images showing examples of color bleeding, direct vs indirect lightning and caustics are presented in figure 1.

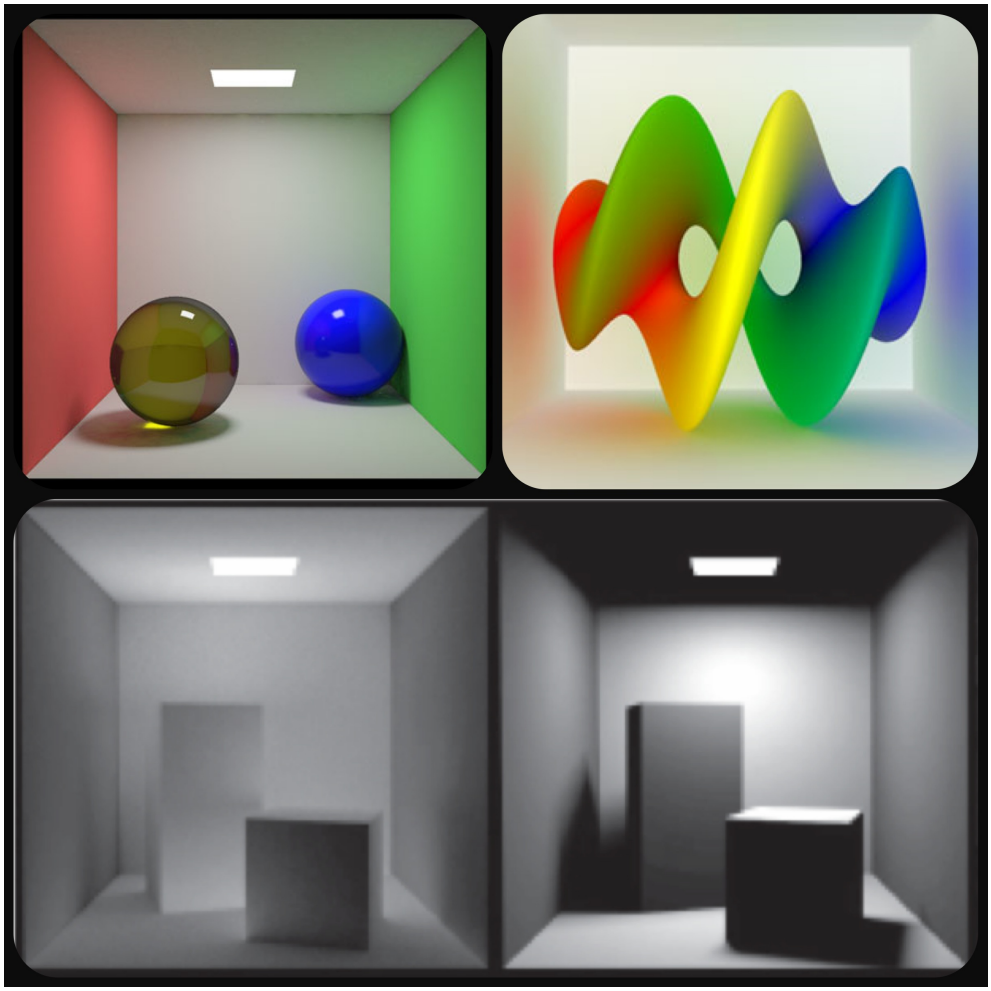

Figure 1: Upper left: caustics, upper right: color bleeding, down left: indirect lightnig, down right: direct ligtning

1.1.3 Whitted ray tracing

The general idea of ray tracing is to produce 2D images from a 3D scene by simulating how rays of light travel.The term 'ray tracing' has over the years been used for different algorithms. Ray tracing associated with global illumination usually implies 'stochastic ray tracing', that allows one to compute a full solution to the rendering equation. Whereas the traditional ray tracing algorithm is often refered to as 'classic ray tracing' or 'whitted ray tracing.

One important difference between light in reality and ray tracing is that rays are tracked backwards. This means rays are tracked from the observer backwards into the scene and taking into account the information from the light at the end. The biggest advantages of this is that all light rays coming from each light source does not have to be simulated, since most of them miss the observer anyway.

One ray per pixel is launched from the observer through an imageplane. The imageplane is a 2D version of the render target. The ray tracer determines the nearest intersection point between the launched ray and the scene geometry, see figure 3. From the intersection point a second ray is launched to simulate:

- Reflection - A reflected ray continues on in the direction it is reflected off the surface and intersects with the closest object in its path and that is what is seen in the reflection.
- Refraction - The ray is bended as it passes into or out of an object.
- Shadow - To avoid tracing all rays in a scene, shadow rays are used to test if a surface is visible to a light.

All these rays are then combined to set a color, and apply it to the pixel the ray was casted through. If a ray hits a surface, and this surface faces a light source, a ray is traced between this intersection point and the light. If the ray hits a dark spot the ray 'dies' and stops tracing. Traditional ray tracing includes only the first ray casted. What Whitted proposed, was to continue this process and launch additional rays from the first intersection point. This whole new layer of ray tracing added huge amounts of realism to ray traced images.

1.1.4 Radiosity

The concept behind radiosity is to compute the average light intensity per area unit at smaller sections of objects, called patches. To calculate the average radiosity in the scene, the algorithm uses stored illumination at the surface of each object. The outgoing light is calculated as a function of incoming light and the amount of radiated light, from each patch. Light arriving directly from the light source is calculated in the first iteration in the radiosity algorithm. In the second iteration, light from other objects is added where the initial light has bounced. This results in an effect called 'color bleeding', which is indirect light that causes transfer of color between nearby objects. Radiosity is a less popular metod compared to ray tracing, and one of the main reasons is that it is more difficult to parallelize. When dealing with ray tracing, each pixel is independent of neighbouring pixels and that is a great advantage. Because of the fact that todays computers and CPU:s have multiple cores, it is possible to allow the cores to handle computations simultaneously, thus speeding up the process.

1.1.5 Monte Carlo ray tracing

Monte Carlo ray tracing is based on the Monte Carlo integration method that descibes a stochastic behavior. It is an extension from the Whitted ray tracer.

When a ray hits an object the new ray-direction gets determined by a stochastic process. This involves using a hemisphere to calculate the new direction. A method called Russian Roulette is used to determine if the ray gets reflected, transmitted or absorbed by the object. The same method also takes the objects material into consideration when determining this stochastic process.

A drawback of the Monte Carlo method is that by letting the rays reflect in random angles, noise gets introduced. If only one ray is reflected, it might hit the lightsource and the pixel next to it might hit a wall, which will create a large differece in pixel-color value (noise). By increasing the amount of rays per pixel, an avrage color value can be calculated, thus reducing noise.

The Monte Carlo method is unbiased, since the stopping conditions for each ray is calculated using a random process rather than setting it manually.

This reminds of how light actually behaves in real life, which results in more realistic computer generated images.

1.1.6 Two-pass rendering

The idea behind two pass rendering is to combine radiosity and ray tracing in order to achive a more realistic representation of light transport. By combining the best of the two it is possible to represent four different forms of light transport:

- Diffuse - diffuse
- Diffuse - specular
- Specular - diffuse
- Specular - specular

The first pass of the two-pass rendering approach is based on enhanced radiosity, while the second pass involves enhanced raytracing. In summary, the two-pass rendering approach is very expensive, since it adds the cost of both radiosity and ray tracing. Even though the combinations enhances the result, there are still many light transport approximations remaining. On the other hand, the two pass rendering approach concludes a promising result. It produces convincing effects and works well for scenes consisting of a small number of reflecting/transmitting objects. Also, it is capable of computing high-quality images when combined with other methods.

1.1.7 Photon Mapping

The basic idea behind photon mapping is to decouple the scene representation from its geometry while storing the illumination information in a structure called: *photon map*. Photon mapping is a two pass method that firstly builds the global photon map by launching and tracing photos from the light source. In the second pass the scene is rendered using the information stored in the photon map created in the first pass. Below, a brief explanation of the the two passes in photon mapping is presented:

Pass 1: Light emission and photon scattering. The first pass creates and launches the photons from the light source into the scene. Photons propagate flux and a light source with a higher intensity will produce more photons. Each photon have a direction that is randomly chosen based on what type of

light source it was launched from. The photons then propagate throughout the scene and collide with different objects. When a photon hits an object, it can be transmitted, absorbed or reflected.

Pass 2: Radiance estimate and rendering. The second pass renders the the scene using a modified Monte Carlo ray tracer with help from the information stored in the photon map. By having a photon map, that is detached from the geometry in the scene, it is possible to render very complex scenes with global illumination. Rays are launched from the camera into the scene, and when a ray hits a point P on a surface, the illumination of nearby photons are collected to calculate the radiance contribution at P, seen in figure 2. By defining a small sphere around the point P and consider all photons enclosed by the sphere together with their indicent direction d, it is possible to decide which photons contribute light.

The concept of photon mapping was introduced by Henric Wann Jensen and it is an efficient alternative to methods like Mote Carlo ray tracing. Jensen has been developing and optimising the concept to handle for instance volume caustics. Interested readers are cited to Jensens article: *Photons with participating media* [3] for further reading.

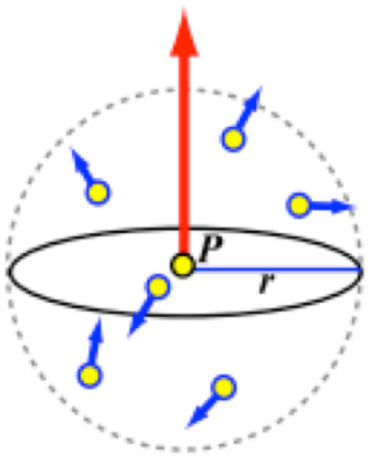

Figure 2: Radience esimate in photon mapping

2 Background

The programming language used to write this Whitted ray tracer was C++. When running the program the scene is presented and saved in a bitmap-file (BMP). The scene used in this program contains spheres of different sizes and material properties.

2.1 Denotations

Vectors and matrices will in this report be denoted as: $\vec{X}$ while scalars and vertices will be denoted as: $\mathbf{x}$

2.2 Cornell Box

The scene setup used is a Cornell box [6]. It is a widely used 3D test model to determine the accuracy of rendering software. The scene consists of a ceiling, a floor and 3 colored walls. Different objects are then placed in the scene in order for it to be compared with the original Cornell box photograph.

2.3 Scene, camera and image plane

Each object is defined implicitly and is assigned a position in relation to the world coordinates. All object have assigned material properties such as color, reflection constant, refraction index, refraction amount and specularity. Each point light source is given a position in the 3D scene. The camera is assigned a position and determines the FOV - based on the distance to the image-plane located in front of the camera, see figure 3. Field of view determines how much of the scene is visible. The camera has no physical appearance, it only determines from where the rays are starting.

2.4 The rendering equation

The rendering equation was introduced by J.T Kajiya in 1986, it consists of five parts and is used to calculate the total light at a point $\mathbf{x}$ in a direction θ.
A simplified version of the rendering equation can be written like eq. 2.4.1 where L_e is the emitted light (outgoing radiance) and L_r is the reflected light.

$$L(x \to \Theta) = (L_e \to \Theta) + (L_r \to \Theta) \tag{2.4.1}$$

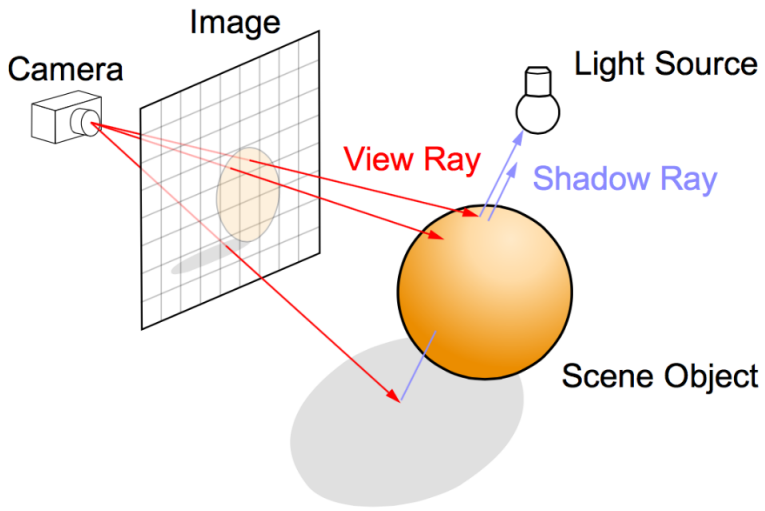

Figure 3: Scene with camera, imageplane, lightsource and object [2]

Because of the fact that the term L_r represents reflected light from all direction of the hemisphere, equation 2.4.1 can be rewritten as an integral, seen in 2.4.2

In equation 2.4.2 L_o is the outgoing light. L_e Is still the emitted light. The integral integrates over all directions ψ in the hemisphere around the point $\mathbf{x}$. f_r is the bidirectional reflectance distribution function (BRDF) which describes how the light is reflected at an opaque surface. L_i is the incoming light in a point $\mathbf{x}$. Finally the cosine term attenuates the incoming light based on the normal at point $\mathbf{x}$ together with incoming light direction.

$$L_o(x \to \Theta) = (L_e \to \Theta) + \int f_r(x, \psi \to \Theta) * L_i(x \leftarrow \psi) * cos(N_x, \psi) d\omega_\psi \tag{2.4.2}$$

2.5 Intersections

In order to determine if any object has been hit by the launched ray, intersections between them has to be checked. An implemented algorithm determines which intersection method to use, based on what type of object

the ray intersects with. First the distance from the observer to all intersection points in the scene is calculated. Secondly it is determined which object is closest to the observer. This is the object shown. The different intersections methiods are presented below.

2.5.1 Ray-plane intersection

A ray has an origin **O**, and a direction **D**. The equation describes a general point on a line. This is called the ray-equation, see 2.5.1.

$$R(t) = O + t * D, t > 0 \tag{2.5.1}$$

Equation 2.5.2 describes an infinite plane which is defined by its normal and with a distance **d** from origo.

$$N * P + d = 0; N * R(t) + d = 0 \tag{2.5.2}$$

By inserting the ray-equation 2.5.1 into the plane-equation 2.5.2, **t** can be solved in equation 2.5.3. **t** represents the distance and **t** bigger than zero indicates that an object has been hit.

$$t = -\frac{O * N + d}{D * N} \tag{2.5.3}$$

2.5.2 Ray-shpere intersection

In 2.5.4 a point **P** lies on the surface of a sphere with centerpoint **C** and radius **r**.

$$(P - C) * (P - C) = r^2 \tag{2.5.4}$$

The P in the ray-equation 2.5.1 gets substiuted and makes it possible to calculate t.

$$(O - t * D - C) * (O - t * D - C) = r^{2} \tag{2.5.5}$$

2.6 Equations regarding reflections and refractions [4]

Equation 2.6.1 for reflected ray **r**:

$$\vec{r} = \vec{i} - 2 * cos(\theta_i) \tag{2.6.1}$$

Equation 2.6.2 for refracted(transmitted) ray **t**:

$$\vec{t} = \frac{n1}{n2}\vec{i} - (\frac{n1}{n2}cos(\theta_i) + \sqrt{1 - sin(\theta_t)^2})\vec{n} \tag{2.6.2}$$

$$cos(\theta_i) = \vec{i} * \vec{n} \tag{2.6.3}$$

$$sin(\theta_t)^2 = (\frac{n1}{n2}) * (1 - cos(\theta_i)^2) \tag{2.6.4}$$

Snell's law is shown below in equation 2.6.5:

$$n_1 sin(\Theta_1) = n_2 sin(\Theta_2) \tag{2.6.5}$$

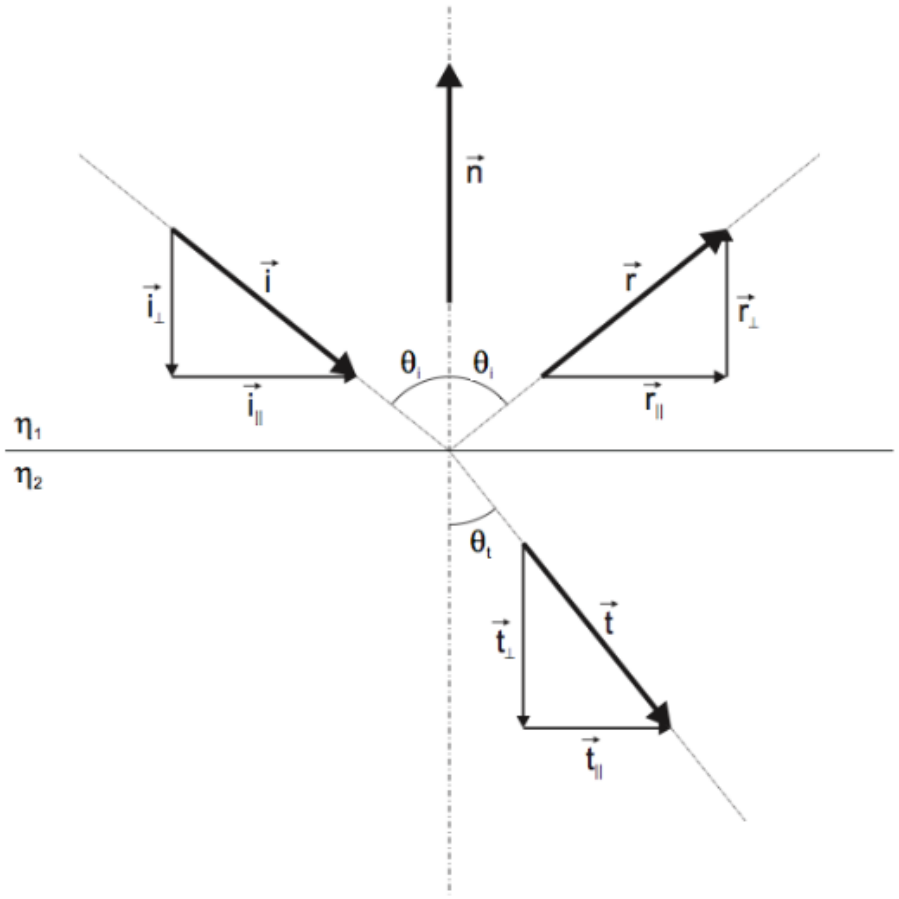

Figure 4: Transmission $\vec{t}$ and perfect reflection $\vec{r}$ [4]

2.7 Shadows

The Whitted ray tracer implemented have so called hard shadows. These are calculated by shooting a ray from the eye and calculating the closest

intersection point with an object. We then, from this intersection point, shoot a shadow ray towards the light and check if there is any intersection occuring between them. See figure 3. If infact one occurs, that pixel will be set to black.

2.8 Reflections

The reflecting rays are only being calculated when hitting specular objects. If the object is diffuse, the ray will be terminated. What needs to be kept in mind is that the ray looses importance (energy) every time it hits an object depending on how specular it is. If the energy gets too low, the iteration stops. We only use the perfect reflection (figure 4) since we have implmenteted a Whitted ray-tracer.

2.9 Refractions

When calculating refracting rays there are a few more things to handle. First of all checking if the object is transparant. If it is, the right refraction coefficient need to be applied to the chosen material. This means that we need to use snells law2.6.5 to calculate how the ray is bending into the transparent medium. Now comes the tricky part. When the ray is inside the object , the normal has to be flipped, the refraction coefficients swapped and before leaving the object, reflect another ray inside.

2.10 The algorithm for the function raytrace(...)

1. **if**(Num of iter <max iter OR importance >min importance)
2. **then**
3. return
4. Calculate the closest intersection point
5. Calculate the normal in the intersection point
6. Calculate refraction
7. **if**(intersected object refraction amount >0)
8. **then**
9. **if**(inside object)

```
10.             then
11.                 turn normal
12.                 switch refraction coeficient
13.             Calculate new refraction ray
14.     Calculate reflection
15.         if( intersected object reflection amount >0 )
16.         then
17.             Calculate new reflective ray
18.     for( all lights in the scene )
19.         Calculate the light direction
20.         Calculate the shadowray
21.         if ( intesection point not in shadow )
22.         then
23.             Calculate lambert - Diffuse shading
24.             Set lambert color
25.             Calculate specular - Blinn phong
26.             if( Blinn not equal to 0 )
27.             then
28.                 Set specular color
```

3 Results and benchmarks

The tests have been performed on a laptop with an Intel i7-3610M 2.38Hz 8-core processor with 8GB 1600MHz of RAM.

Table 1: Recursion depth benchmarks

Recursion depth	Time (seconds)
1	16
5	16
10	16
20	16

Table 2: Resolution - benchmarks

Resolution (pixels)	Time (seconds)
100x100	1 s
300x300	6 s
600x600	25 s
1000x1000	78 s
1200x1200	122 s

Table 3: Material properties - benchmarks

Resolution (pixels)	Time (seconds)	Scene
500x500	16 s	Three totally diffuse sphere
500x500	17 s	Three totally reflective spheres
500x500	18 s	Three totally refractive spheres
500x500	18 s	3 spheres: reflective, refractive and diffuse

3.1 Benchmarks

From Table 1 it is possible to conclude that the number of times the ray gets reflected and refracted does not affect the render time significantly. The images was rendered with the resolution 500 x 500 pixels.

The information in Table 2 indicates that the render time increases suffiently with the number of rendered pixels.

From Table 3 there is a small indication that refraction and reflection extends the render time. In summary increasing the resolutin and/or the number of objects in the scene will result in longer render time. Especially when it comes to reflective and refractive objects. This is not suprising since alot more computations will need to be performed.

3.2 Results

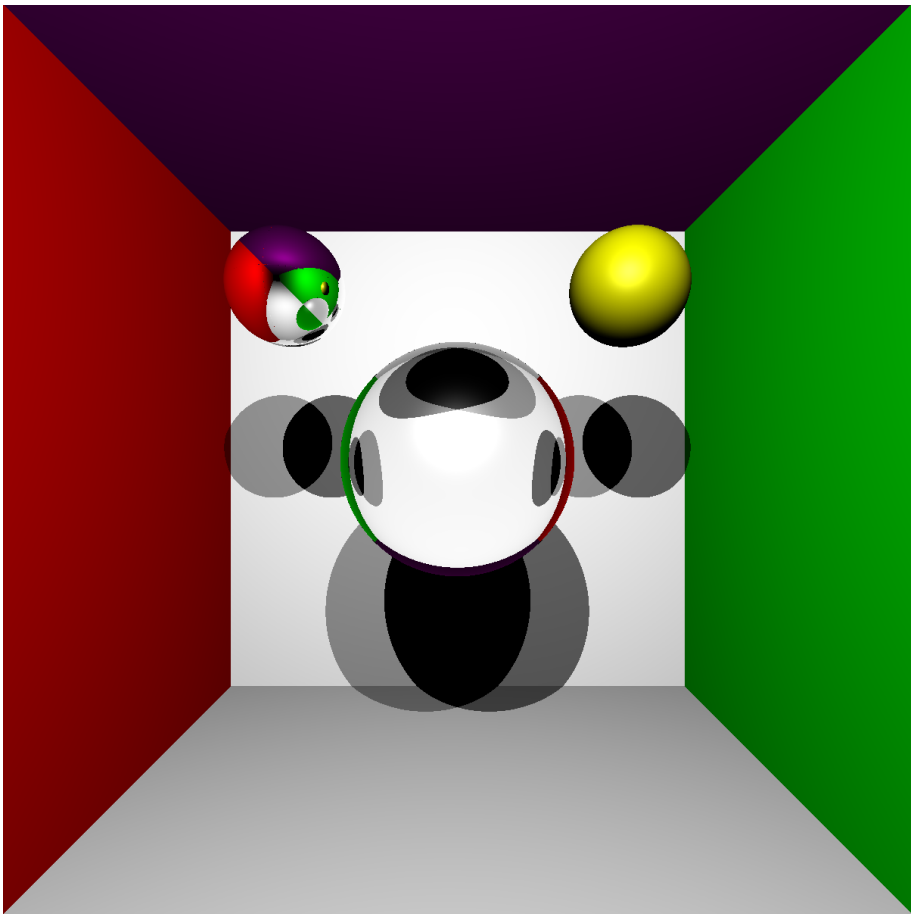

Figure 5: An image rendering with the Whitted ray tracer. Resolution 1200x1200. The scene contains of two point ligh sources with slightly different position, hence the displacement of the different shadows seen on the back wall.

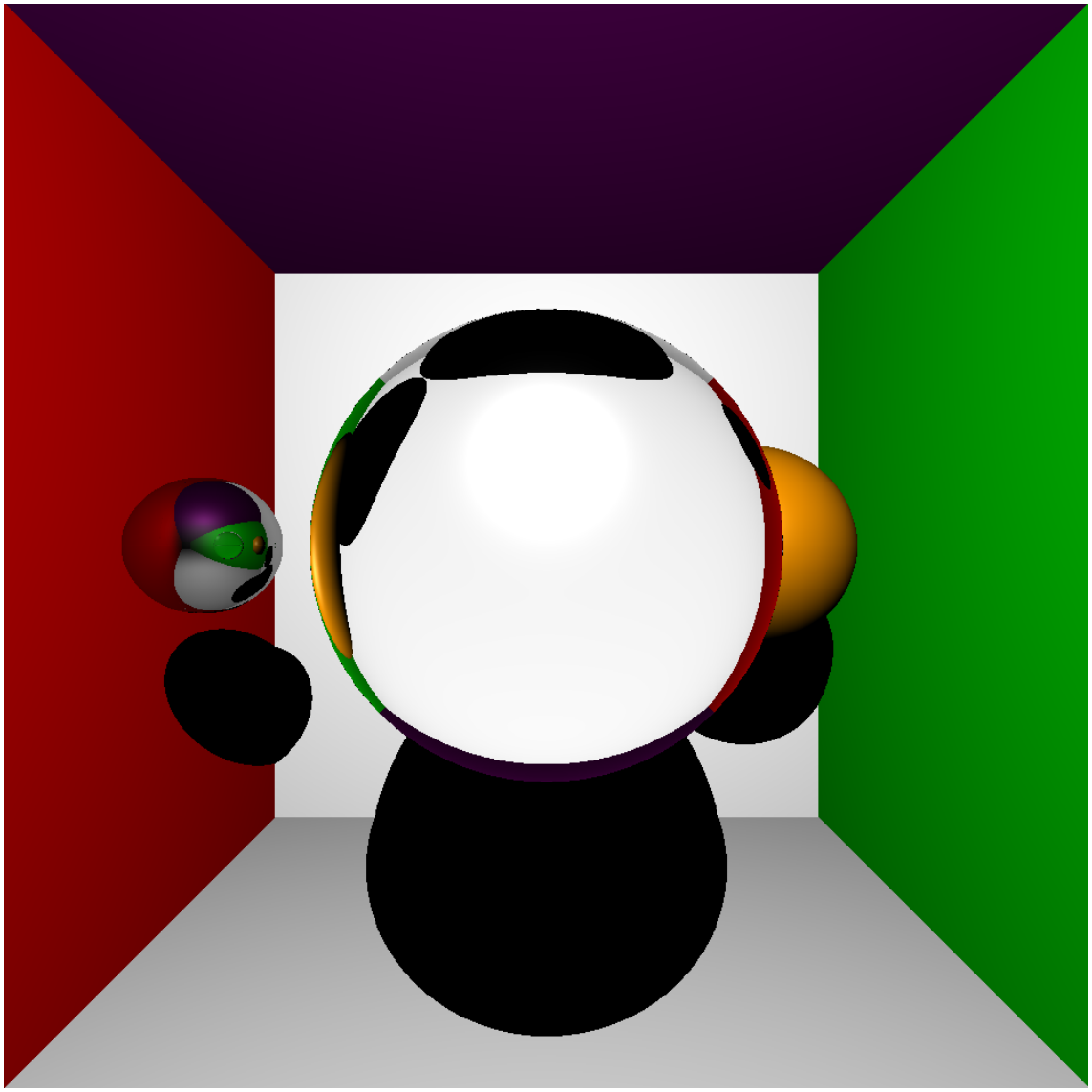

Figure 6: An image rendering with the Whitted ray tracer. Resolution 1200x1200. The scene contains only one point light source, resulting in hard shadows.

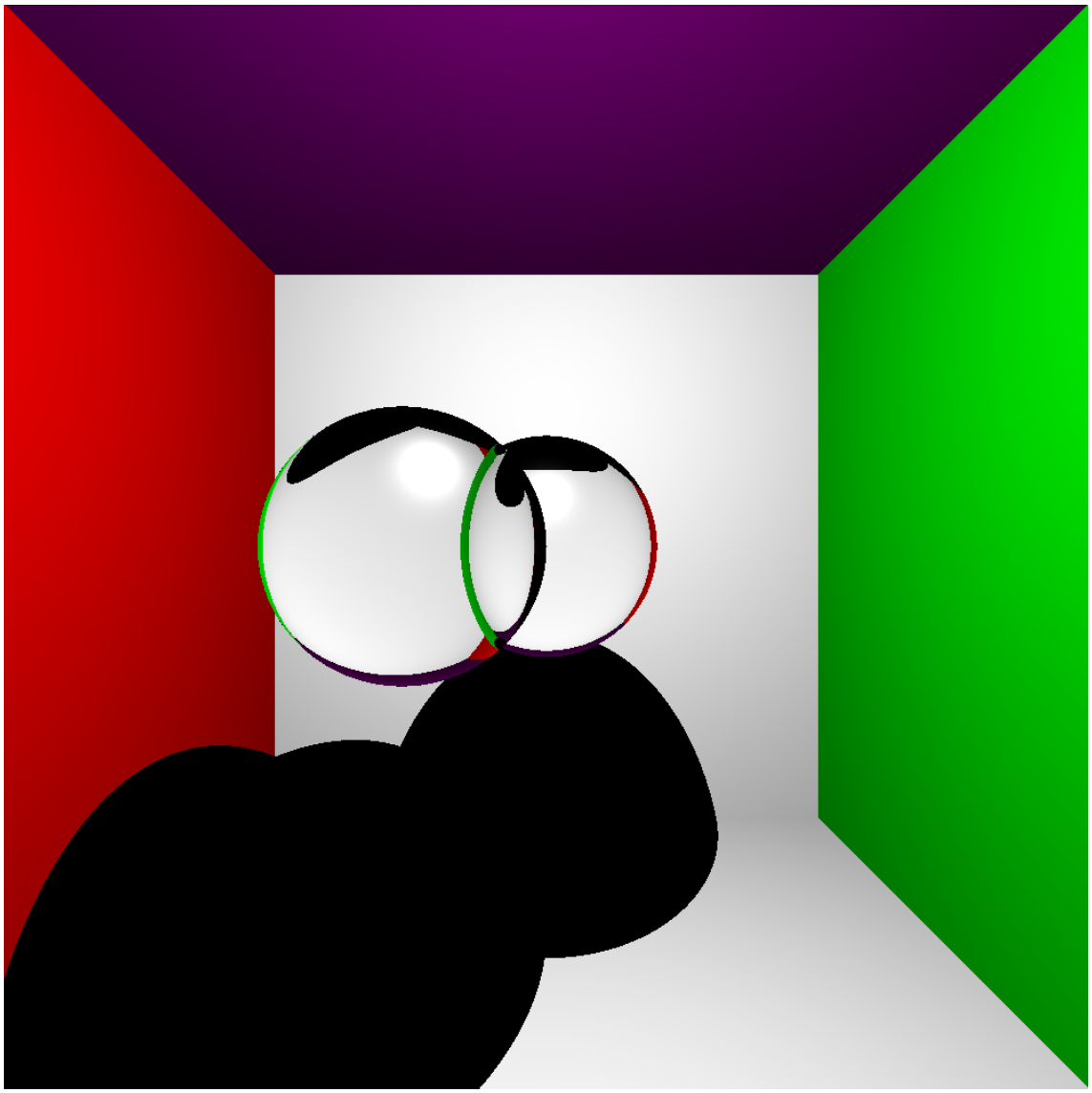

Figure 7: An image rendering with the Whitted ray tracer. Resolution 1200x1200. The scene consists of one point light source together with two intersecting glass spheres (refraction index: 1.5).

4 Discussion

Implementing a Whitted ray tracer is a good way to gain knowledge about the basic concept of light transportation within a scene. A fully functional ray tracer requires a framework that can handle:

- intersections between rays and different object
- camera and scene definitions
- material properties

Our goal was to implement a Whitted ray tracer and fully understand the concept behind it. We chose the simpler alternative rather than implementing a Monte Carlo ray tracer since we did not have any previous knowledge regarding the implementation. So this project has been instructive but still very challenging, despite the fact that we had previous knowledge about the basics in computer graphics.

One of many improvements of the program could be to extend the existing framework to a Monte Carlo ray tracer. This would include an extension of the path tracer with Russian roulette sampling in order to obtain a unbiased stopping condition for the propagating rays.

The material class can also be extended with a BRDF to enable correct ray-surface interactions.

Since we only use one ray per pixel, our result suffers from aliasing. One way to reduce the amount of aliasing effects is to implement sumpersampling, by subdividing each pixel and then select points on the imageplane each corresponding to one subpixel.

Multiprocessing support would also be an step forward in order to improve the rendering time. By allowing the cores to run calculations simultaneously, the process could be parallelised and sped up drastically. Another component that increases efficiency is optimising the intersection calculations. This can be done by implementing a tree structure that reduces the number of unnecessary intersections.

After having implemented a Monte Carlo ray tracer, one further improvement would be to extend it by combining it with photon mapping. This

would speed up the render time and simultaneously give better result with less noise.

References

[1] Watt Alan, *3D Computer Graphics, third edition.* Adison Wesley, 2000.

[2] *http://tinyurl.com/bu7qpgd.*
Visited: 2013-01-29

[3] Henric Wann Jensen *Photon mapping using participating media http://dl.acm.org/citation.cfm?doid=280814.280925*
Visited: 2013-01-29

[4] *http://tinyurl.com/cy4vxde*
Visited: 2013-01-29

[5] *Philip Dutre, Kavita Bala, Philippe Bakaert. Advanced Global Illumination, Second edition.* A K Peters, 2006.

[6] *http://design.osu.edu/carlson/history/PD Fs/goral.pdf*
Visited: 2013-02-01