**Photon Mapping and Approximations**

**Jianyang Zhang Xuyang Wang Xuan Zhuang Yufei Zhang**

# 1 Introduction

The photon map method is an extension of ray tracing. Unlike simple ray tracing, photon mapping is more realistic. And it is capable of handling diffuse inter-reflections and caustics. To conveniently show the advantages of photon mapping, we design this project, which implements basic ray tracing and photon mapping separately in Cornell Box.

Our report is divided into five parts. In the second part, background knowledge of photon mapping will be introduced. Next, we will show the concrete implementations of our project. Then, we will summarize our work. Finally, we will describe some possible improvements.

# 2 Background

Photon Mapping is a two-pass global illumination algorithm. The first pass is photon tracing. Photons are emitted from light sources into the scene, and stored in a photon map. The second pass is rendering, using stored photon map to approximate the pixels’ radiance by statistical techniques.

Compared with Monte Carlo ray tracing, the main benefit of photon mapping is efficiency, which means its computation time is less for implementing the same result. And it offers a solution to diffuse inter-reflections and caustics.

# 3 Implementation

## 3.1 Models

In this project, our model is based on Cornell box. There are three spheres inside it. Two of them has a reflection property. The other one has a refraction property.

## 3.2 Light Sources

### 3.2.1 Diffuse Point Light

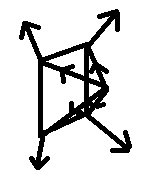
To simulate a point light, we emit photons in all directions uniformly. (Figure 1)



*Figure 1: Diffuse point light*

### 3.2.2 Square Light (two types)

TYPE 1: To simulate a square light, we put a point light slightly in front of a square object which is close enough to the wall. Then, we randomly emit photons from this point light to hit the surface of the square. As a result, the reflecting photons can be regard as the photons which emit from this square object (square light source). (Figure 2)



*Figure 2: Square light type 1*

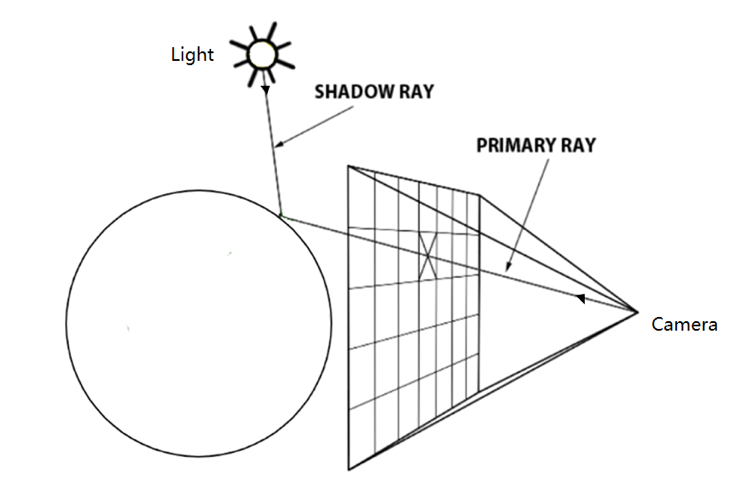
TYPE 2: We choose a random position of the square area and pick random direction in the hemisphere above this position to emit photons. Repeat this action several times to simulate a square light. (Figure 3)



*Figure 3: Square light type 2*

## 3.3 Basic Ray Tracing

For each pixel, there is a primary ray shot from camera through it. The shot ray may be reflected, transmitted, or absorbed. The details of these three processes will be introduced in part 3.4.2. We record the object the shot ray is finally intersected with. Then, we send a shadow ray from light to the intersected point. If the recorded object is the closest object shadow ray hits, the hit point is illuminated (Figure 4); otherwise, it is shadowed. Therefore, the color of pixel is confirmed.



*Figure 4: Basic ray tracing*

## 3.4 Photon Mapping

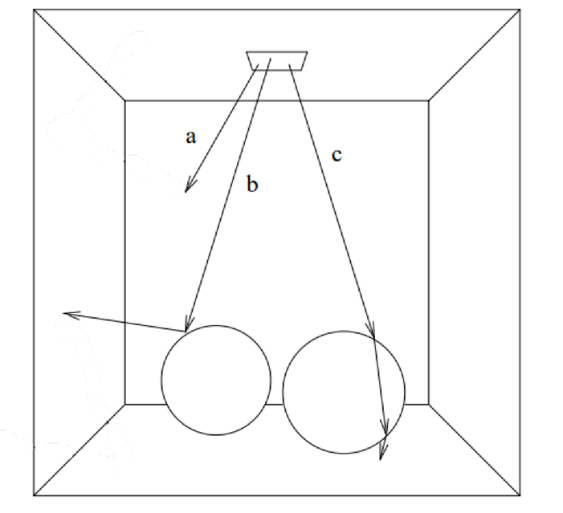
### 3.4.1 Constructing the Photon Maps

To construct the photon maps, we use the light sources models mentioned in 3.2 part to emit totally 4096 photons. For each photon, it can bounce three times. For each photon bounce, we store the location of this bounce, direction from which this photon hit the object, energy which this photon emit in this bounce (relating to the times the photon has bounced) and we also compute a single corresponding shadow photon.

Photon mapping is enabled simply by replacing the final lighting step of ray tracing with a photon gathering step in which, for any scene point being rendered, photons are integrated over a fixed-size area.

### 3.4.2 Photon Scattering

Each photon can only be reflected, transmitted or absorbed after hitting objects in the scene. (Figure 5)



*Figure 5: (a) Absorbed (b) Reflection (c) Refraction*

Equation of computing reflection vector is:

Where *L* is a unit vector of incident ray, *N* is a normal vector of reflexible object.

Equation of computing refraction vector is complicated. It is shown as following:

Where *L* is a unit vector of incident ray, is relative index of refraction, *N* is a normal vector of refrangible object,is angle of incidence, is angle of angle of refraction.

In this equation, we can easily get and . But we do not know the value of . Therefore, we need to get it by and .

According to refraction law, . Therefore, .

### 3.4.3 Photon Storage

We simply use a 3D array to store the photon information. Here is the detail:

*Photons[object\_id][photon\_num][3]*

object\_id --- index of the objects

photon\_num --- current number of the photons which hitted this object

0 --- location of the photon hit on this object (object\_id)

1 --- direction of direction from which this photon hit the object

2 --- energy (relating to color) which this photon emit in this hit.

### 3.4.4 Shadow Photon

1)To compute the corresponding shadow photon after each bounce, we first create a shadow photon with negative energy.

2)Then, we start just beyond the intersection location (bounce location) in the direction which is the same as the photon flies before this bounce to emit this shadow photon.

3)After that, we store this shadow photon information in the method of part 3.4.3

## 3.5 Rendering

### 3.5.1 Computing Pixel Color

Step 1 - Ray tracing

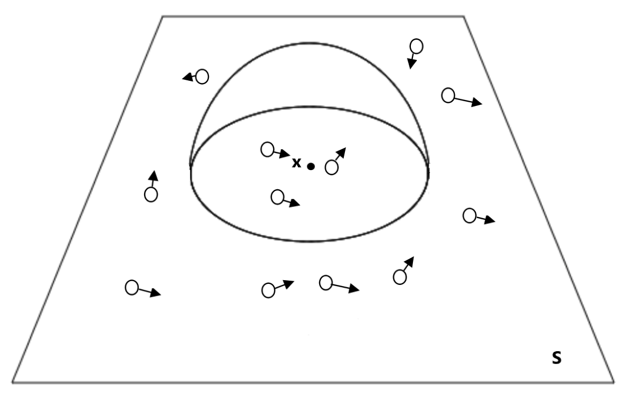
For each pixel on screen, we shoot a ray from the camera (Focal length = 1.0) through this pixel into the scene (allow the reflection from the specular object). For the final object the ray hit, we compute the corresponding color of this location. To compute color, we need to use the photon maps we constructed in part 3.4 and gather the photons in a fixed-size area around this location to compute the pixel color.

Step 2 - Gathering the photons

To compute the radiance efficiently, we use an estimating algorithm based on photon map computed before. For the radiance of the point **x** at the surface **s**, we use all photons within a fixed size sphere (whose radius is **r**) area which is center is point **x**. Based on the distance **d** between point **x** and each photon, we use a coefficient to adjust the energy of each photon. Where k is a filter constant to enhance the sharpness of estimate.

The filtered radiance estimate can be expressed as:

Where is the radiance of each photon and is the angle between norm and the ray of photon. We will collect the radiance of photons near the point **x** and divide the area of sphere to get an average radiance as the estimate. Then we can easily compute the color based on this estimate. (Figure 6)



*Figure 6: Gather photons*

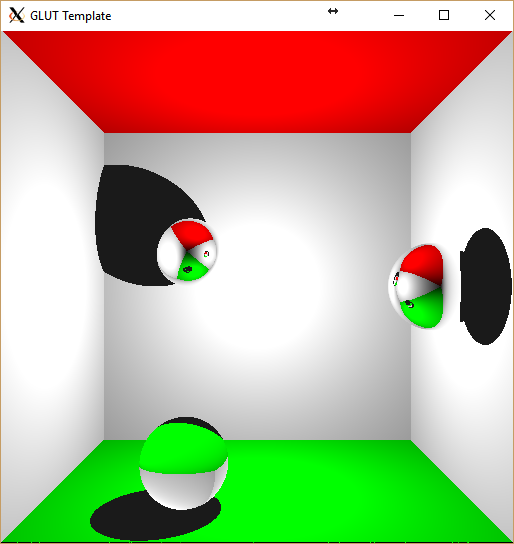
### 3.5.2 Display “Trick”

For user-friendly interface, we need to display something immediately instead of waiting for long time to compute the final image. As a result, we implement a “trick” to display a rough image immediately by render pixels out of order with increasing resolution: 2x2, 4x4, 16x16... 512x512.

# 4. Conclusion

In this project, we implement basic ray tracing, photon mapping. The following figures are all results. As we can see from results, photon mapping is more realistic and solve the caustic problem.

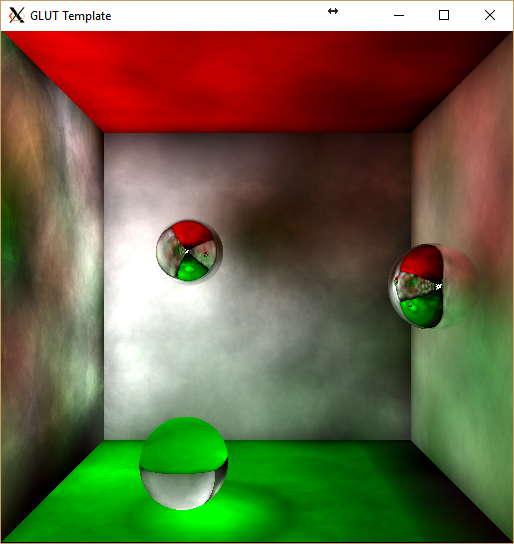
## 4.1 Basic Ray Tracing Result:



*Figure 7: Basic ray tracing result*

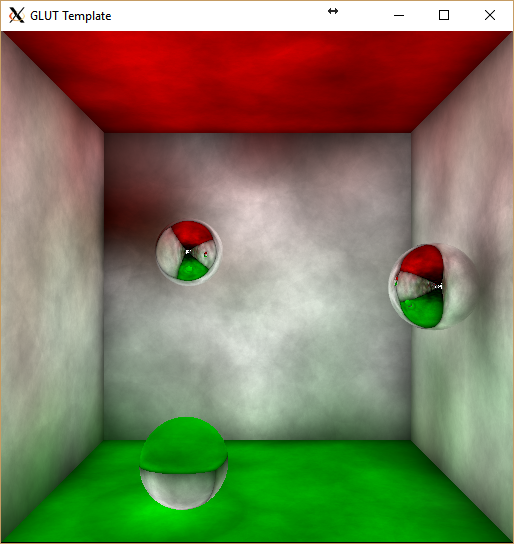
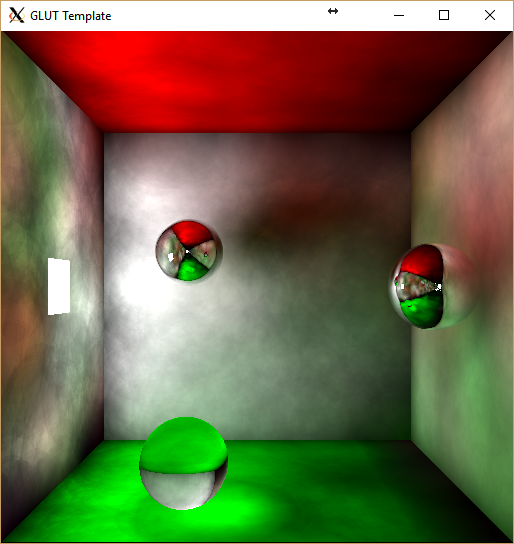
## 4.2 Photon Mapping Result:

1) Diffuse Point Light Result:



*Figure 8: Diffuse point light result*

2) Square Light Results:



*Figure 9-1: Square light type 1 result Figure 9-2: Square light type 2 result*

# 5.  Improvement

We can use KD-Tree to store the photon information and the time complexity can reduce to O(log n) if it is a balanced tree.

# References

# [1] Jensen, H. W. (1996). Global illumination using photon maps. *Rendering techniques*, *96*, 21-30.

# [2] <https://graphics.stanford.edu/courses/cs348b-01/course8.pdf>

# [3] https://www.scratchapixel.com/lessons/3d-basic-rendering/introduction-to-ray-tracing/implementing-the-raytracing-algorithm