Multiplexed Metropolis Light Transport

Chen-Yuan Hsu

Master of Science,

Computer Animation and Visual Effects

August, 2014

Contents

List of Figures

Abstract

There is no doubt that global illumination rendering techniques are one of the major research areas in computer graphics. The goal of these rendering algorithms is to generate photo-realistic images. Therefore, to achieve this, indirect illumination could be the most important issue by reason that it is not straightforward to efficiently simulate light transport.

Multiplexed Metropolis Light Transport is one of the up-to-date rendering algorithms to significantly solve indirect illumination problems. Furthermore, the implementation of this algorithm is easier than other complicated Markov chain Monte Carlo rendering algorithms, and its performance is still effective.

In this project, multiplexed Metropolis light transport has been successfully implemented and extended from the rendering engine which was written for previous assignments. The quality of the final rendered images will demonstrate the efficiency of this algorithm for solving indirect illumination problems.

Keywords: global illumination, Metropolis light transport, multiple importance sampling

Acknowledgements

I would like to give my deepest appreciation to Mr. Mathieu Sanchez, Dr. Ian Stephenson and Mr. Jon Macey. Mr. Mathieu Sanchez gave me many suggestions and encouraged me to find solutions by myself. I think that it was very helpful and useful to improve my problem-solving ability. Dr. Ian Stephenson guided me to have critical thinking about my project, so I could realise my project more and ensure I was moving in the correct direction. Mr. Jon Macey taught me several important software development skills. Hence, I could use these skills to achieve my project properly.

Finally, I would like to thank my colleagues and other professors who taught me. You brought me a memorable experience in my life. Wish you have a good luck. Thank you.

Chapter 1

Introduction

In order to render plausible images, there are numerous global illumination rendering algorithms which are based on using Monte Carlo techniques to simulate light transport. In the real world, it can be seen that the environment contains not only direct illumination but also intricate indirect illumination. Hence, the most essential nature for these rendering algorithms could be to efficiently solve indirect illumination problems. In general, it is applicable for several classic global illumination rendering algorithms to achieve this purpose in a simple scene, if the light sources are very obvious. However, in most scenes, the light sources are implicit, and it is particularly difficult to fake their effects when the environment contains strong indirect illumination.

Multiplexed Metropolis light transport Hachisuka *et al.* (2014) is one of the latest rendering algorithms to efficiently solve indirect illumination issues. Its concept is extended from primary sample space Metropolis light transport Kelemen *et al.* (2002) and combines multiple importance sampling with Markov chain Monte Carlo sampling to obtain proper paths for light transport simulation. In terms of solving indirect illumination issues, the quality of the final rendered results using this algorithm are persuasive, and its performance is much more efficient than several classic global illumination rendering algorithms.

In this project, the goal is to implement multiplexed Metropolis light

transport and render high quality images. In terms of implementation, it can be carried out by extending the rendering engine which was written for former assignments.

Chapter 2

Related Work

Over the past few years, it is evident that adopting stochastic methods to solve complex light transport simulation has been a common trend. Path tracing Kajiya (1986) uses randomly sampled rays from the camera position and then traces paths to calculate the light contribution if rays reached a light source. This algorithm is good at rendering outdoor scenes, but it is inefficient to render indoor scenes when light sources are hard to reach. Light tracing Arvo (1986) uses another idea which sampled rays from the light sources and also traced paths to compute the light contribution if the intersection vertices of each path were in the field of view. The key advantage of this algorithm is that it is able to efficiently deal with some natural phenomena such as caustics, but its performance is not effective. This is because many sampled rays can not be seen by the camera, and it will waste time on computing these invisible rays. In addition, it has the same problem in that if the light sources are too hidden, light tracing is not good at efficiently solving indirect illumination problems due to its computation complexity.

Bidirectional path tracing Lafortune and Willems (1993) sampled rays from both the camera and the light sources and then connected path vertices, so it could utilise the benefits from both path tracing and light tracing. Furthermore, Veach and Guibas (1995) proposed another independent bidirectional path tracing rendering algorithm which utilised a sampling technique for variance reduction. The algorithm known as

multiple importance sampling was used to optimally calculate the contribution of each connected path. This sampling technique was excellent to significantly reduce variance. However, although these aforementioned rendering techniques could simulate global illumination well in many situations, they were still inefficient to solve situations which the light sources were too implicit or small. As a result, the artificial noise was obvious, and a large amount of samples were used to reduce this noise.

Metropolis light transport Veach and Guibas (1997) used the method of bidirectional path tracing to construct paths for light transport, and then modified the paths according to some mutation strategies. The whole process was more efficient than bidirectional path tracing, because it could generate better paths which had higher contributions to the image. Primary sample space Metropolis light transport Kelemen *et al.* (2002) presented a new mutation strategy for Metropolis light transport which could be implemented much easier. Gradient-domain Metropolis light transport Lehtinen *et al.* (2013) provided an extension of path-space Metropolis light transport which needed to calculate image gradients and use these gradients to reconstruct the rendered result. The foregoing Metropolis light transport rendering algorithms are able to solve indirect illumination issues much better, but some of them are difficult to implement.

Multiplexed Metropolis light transport Hachisuka *et al.* (2014) extended the framework of primary sample space Metropolis light transport Kelemen *et al.* (2002) and connected multiple importance sampling with Markov chain Monte Carlo sampling. As compared with those more complicated Metropolis light transport rendering algorithms, the implementation of this algorithm is easier, and its performance is comparable or better.

Chapter 3

Technical Background

3.1 The Bidirectional Scattering Distribution Function Models

From Veach Veach (1998), the bidirectional scattering distribution function (BSDF) is a mathematical description to express the reflection or transmittal of the light at a surface. Basically, the BSDF can be considered as a generalisation of the bidirectional reflectance distribution function (BRDF) and the bidirectional transmittance distribution function (BTDF). It is evident that different materials have different light-scattering properties, and there are four major materials used in this project, which are pure diffuse, pure specular, glossy and dielectric materials.

3.1.1 The Pure Diffuse Model

The pure diffuse model is also called the Lambertian material. It represents the incident light being scattered uniformly in all reflected directions, and this material will have the same incoming radiance from all possible directions. Figure 3.1 demonstrates its concept for an incident light ray. This material is very simple and common, and Figure 3.5 (a) shows an example.

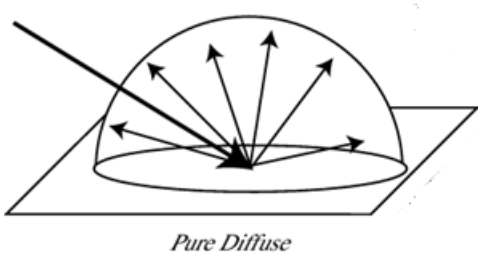


Figure 3.1: *The illustrated concept of the pure diffuse material. The image was taken from Dutre* et al. *(2006)*

3.1.2 The Glossy Model

The glossy model is an imperfect specular reflection material. Simply, it can be considered as a combination of both diffuse and specular reflectance materials shown in Figure 3.2. To simulate imperfect reflection, it can be done by using a cosine-power-weighted formula to calculate the reflected ray Suffern (2007). This reflected ray will be oriented around the direction of mirror reflection. In Figure 3.5 (b), it can bee seen that a glossy material contains both diffuse and specular properties.

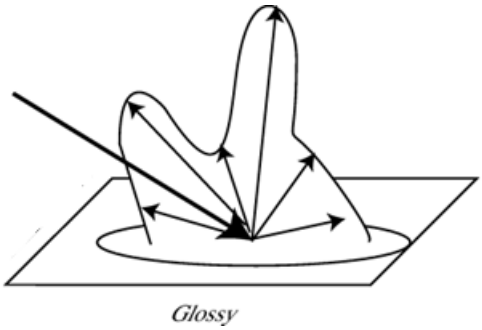


Figure 3.2: *The illustrated concept of the glossy meterial. The image was taken from Dutre* et al. *(2006)*

3.1.3 The Pure Specular Model

The pure specular model follows the law of reflection, which indicates that the incident light and the reflected light have the same angles relative to the surface's normal, and that the directions of the incident light,

surface normal, and reflected light are in the same plane Shirley and Morley (2000). Figure 3.3 illustrates this idea.

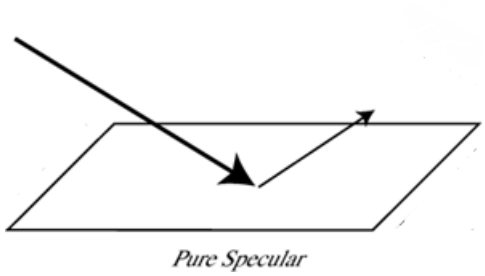


Figure 3.3: *The illustrated concept of the pure specular material. The image was taken from Dutre* et al. *(2006)*

This material is usually used as mirror materials, and an example is shown in Figure 3.5 (c).

3.1.4 The Dielectric Model

A dielectric model is a transparent material which can refract light, and the refracted ray follows the Fresnel equation. The concept is shown in Figure 3.4.

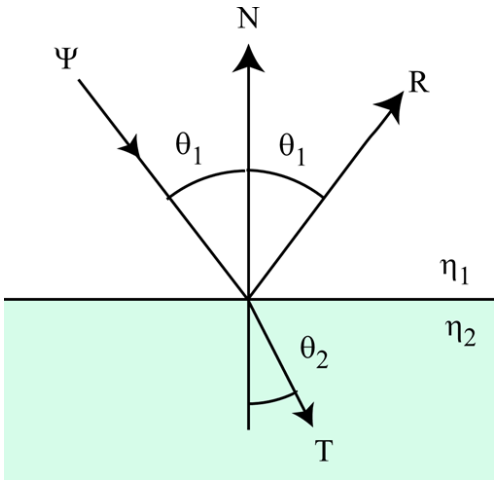
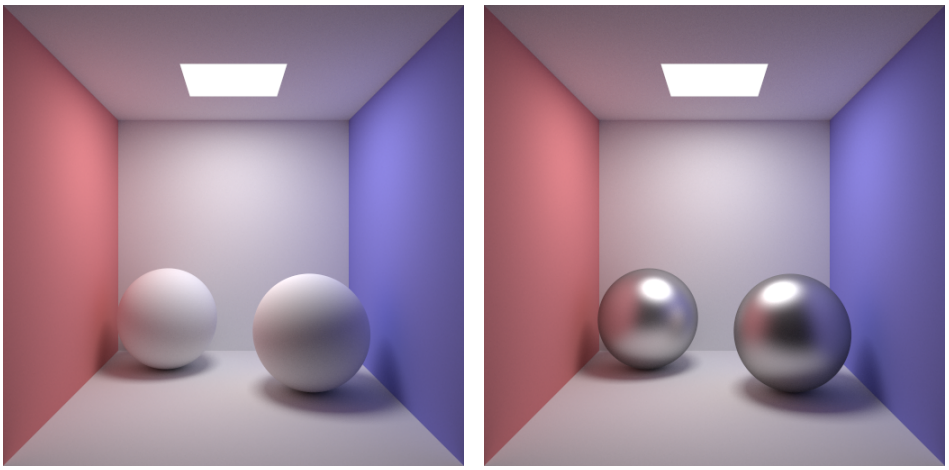


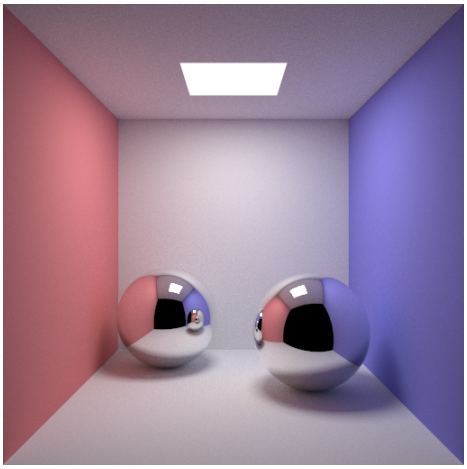
Figure 3.4: *The illustrated concept of the dielectric material.* Ψ *is the incident light ray. N is the surface normal. R is the perfect reflected ray. T is the refracted ray.* η_1 *and* η_2 *are the refractive index of mediums. The image was taken from Dutre* et al. *(2006)*

Furthermore, Schlick developed an exactly right approach to the Fresnel equations Shirley and Morley (2000), and this approach can be used to calculate the change in reflectance.

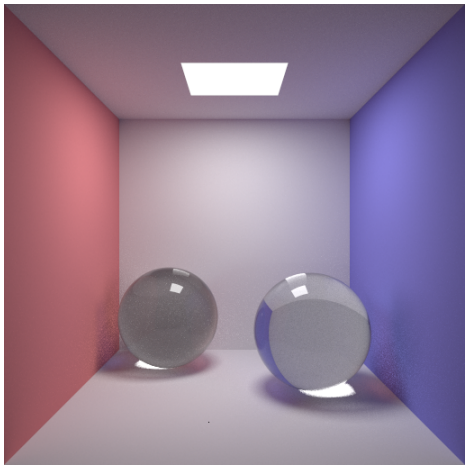


(a) Matte Material

(b) Glossy Material



(c) Reflective Material



(d) Dielectric Material

Figure 3.5: *A simple example shows different BSDF models. These images were rendered using path tracing with the renderin engine.*

3.2 Bidirectional Path Tracing

Bidirectional path tracing Veach and Guibas (1995) is a global illumination rendering algorithm. Its algorithm samples rays from both the camera and the light sources and then traces these rays individually in

a certain depth. After this, each vertex of the camera path and the light path will be connected if there is no obstacle between them. The algorithm is illustrated in Figure 3.6.

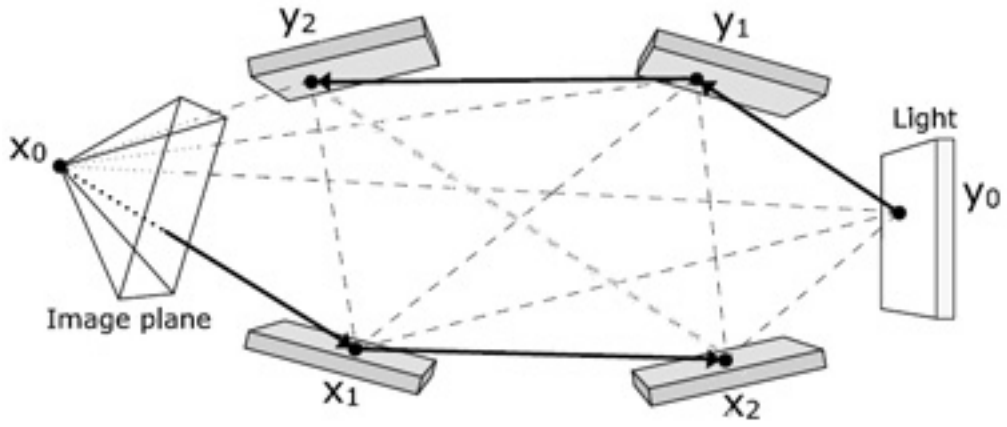


Figure 3.6: *The illustrated concept of the bidirectional path tracing rendering algorithm. The image was taken from Adamsen (2009)*

After connecting the paths, the contribution of each path can be calculated using multiple importance sampling. The way to construct paths in bidirectional path tracing was also used in multiplexed Metropolis light transport for light transport simulation.

3.3 Multiple Importance Sampling

Multiple importance sampling introduced by Veach and Guibas (1995) is a technique used to reduce variance and increase the accuracy of Monte Carlo integration. It combines the estimators from several sampling techniques in an appropriate method. The equation is formulated as below:

$$w_{t,s} = \frac{p_{t,s}^{\beta}}{\sum_{k=0}^{t+s} p_{k,t+s-k}^{\beta}}, \tag{3.1}$$

where $p_{t,s}$ is the probability density of constructing a path with the camera path length t and the light path length s. Moreover, the probability

density $p_{t,s}$ can be evaluated by using the following equation:

$$p(\mathbf{x}_{t\to t+1}) = p_f(\mathbf{x}_{t\to t+1})\frac{|((\mathbf{x}_t - \mathbf{x}_{t+1})\cdot\vec{n}_{x_t})\cdot((\mathbf{x}_t - \mathbf{x}_{t+1})\cdot\vec{n}_{x_{t+1}})|}{||\mathbf{x}_t - \mathbf{x}_{t+1}||^2}, \quad (3.2)$$

where $p_f(\mathbf{x}_{t\to t+1})$ is the probability density of generating the reflected or refracted ray. This sampling technique was used in multiplexed Metropolis light transport to compute the weighted contribution of each connected path.

3.4 Metropolis Sampling

Metropolis sampling was presented by Metropolis *et al.* (1953) for solving sophisticated sampling issues. From their algorithm, integration can be estimated as below:

$$\int g(x)dx \approx \frac{1}{N}\sum_{i=1}^{N}\frac{g(x_i)}{p(x_i)} = \frac{b}{N}\sum_{i=1}^{N}\frac{g(x_i)}{p^*(x_i)}. \quad (3.3)$$

Here $p(x)$ is a probability density function which is proportional to $g(x)$, and $p^*(x)$ is an arbitrary positive scalar function where $p^*(x) \propto p(x)$. In this approach, it means that Metropolis sampling could generate samples based on the unknown function $p^*(x)$ to estimate integrals. In terms of rendering, Metropolis sampling can be used to sample rays according to the image's radiance.

3.5 Metropolis Light Transport

Metropolis light transport Veach and Guibas (1997) first utilised Metropolis sampling technique for rendering, and they applied Equation 3.3 to

the rendering equation and re-formulated the path integral as:

$$I_j \approx \frac{b}{N} \sum_{i=1}^{N} \frac{h_j(\bar{x}_i) f(\bar{x}_i)}{f^*(\bar{x}_i)}, \tag{3.4}$$

where I_j is the j-th pixel's intensity, and $h_j(\bar{x})$ is a non-zero function which is only used to filter the j-th pixel. Furthermore, $f(\bar{x})$ is the path throughput, and $f^*(x)$ is utilised as $p^*(\bar{x})$ in Equation 3.3 which indicates the path throughput's luminosity. In Metropolis light transport, each sample $\bar{\mathbf{X}}_i$ can be acquired by randomly modifying the walk of the previous sample $\bar{\mathbf{X}}_{i-1}$. This random modification is called the mutation, and the new random walk is called a Markov chain, where the new sample is only based on the former sample. If the new path is valid, it can be considered to replace the old path on a basis of the acceptance probability shown as below:

$$a(\bar{x}_i \to \bar{y}) = min \left\{ 1, \frac{f^*(\bar{y}) q(\bar{y} \to \bar{x}_i)}{f^*(\bar{x}_i) q(\bar{x}_i \to \bar{y})} \right\}. \tag{3.5}$$

Here $q(\bar{y} \to \bar{x}_i)$ is a conditional probability density which gives the probability density that $\bar{x}_i = \bar{y}$ given that $\bar{x}_{i-1} = \bar{x}$. Hence, good mutation strategies are important in order to get higher acceptance probability, and then it can focus more on exploring the bright areas of the image.

3.6 Primary Sample Space Metropolis Light Transport

Primary sample space Metropolis light transport Kelemen *et al.* (2002) proposed a simplified method to the Metropolis sampling process. The core of the algorithm uses local mutations for generating a new random sample which is closed to the former one and global mutations called large steps in their paper for generating another totally different sample. Moreover, from their algorithm, the path integral could be simplified as

below:

$$I_j \approx \frac{b}{N} \sum_{i=1}^{N} \frac{\hat{h}_j(\bar{u}_i)\hat{C}(\bar{u}_i)}{\hat{C}^*(\bar{u}_i)}, \tag{3.6}$$

where $\bar{u}_j$ indicates serial random numbers which are mapped to a path $\bar{x}_j$, and $\hat{C}(\bar{u}_j)$ is the path contribution in primary space. Similarly, the acceptance probability in primary space was re-formulated as:

$$a(\bar{u}_i \to \bar{v}) = \frac{\hat{C}^*(\bar{v})}{\hat{C}^*(\bar{u}_i)} \tag{3.7}$$

From their equations, if there is a probability density function $p(\bar{x})$ which is proportional to $f^*(\bar{x})$, the acceptance probability $a(\bar{u}_i \to \bar{v})$ would approach to 1. However, it is not straightforward to find this function, and its mutation methods would be more restricted compared to Veach and Guibas (1997).

Chapter 4

Multiplexed Metropolis Light Transport

Hachisuka *et al.* (2014) proposed multiplexed Metropolis light transport (MMLT), and they mentioned that I_j can be estimated by taking N_t samples $\bar{x}_{t,i}(t = 1...M, i = 1...N_t)$ with a distribution based on $p_t(\bar{x})$. As a result, I_j can be re-formulated as:

$$I_j \approx \sum_{t=1}^{M} \frac{1}{N_t} \sum_{t=1}^{N_t} h_j(\bar{x}_{t,i}) w_t(\bar{x}_{t,i}) C_t(\bar{x}_{t,i}), \tag{4.1}$$

where $w_t(\bar{x})$ is a weighting function, and $\sum_{t=1}^{M} w_t(\bar{x})$ is equal to 1. M is the number of probability density functions. t is the number of the vertices of the camera path, and s is the number of the vertices of the light path. Hence, the total path length k is equal to $(s + t - 1)$.

Furthermore, they extended serial tempering Marinari and Parisi (1992) Geyer and Thompson (1995) to their algorithm's framework. Therefore, in their algorithm, the acceptance probability a would be computed using the following equation:

$$a((\bar{u}, t) \to (\bar{v}, t^{'})) = \frac{\hat{w}_{t'}(\bar{v}) \hat{C}^{*}_{t'}(\bar{v})}{\hat{w}_t(\bar{u}) \hat{C}^{*}_{t}(\bar{u})}. \tag{4.2}$$

MMLT was constructed on the framework of the primary space serial tempering. The major feature of their method is that another Markov chain status is utilised in the algorithm in order to decide how to map the remainder of the Markov chain statuses to a path. Hachisuka *et al.* (2014) showed that it is more efficient while using an additional independent Markov chain for every path length k to save statuses. This is because various path lengths must have their fundamental contribution to various integrals, and this rendering algorithm can focus more on computing the path lengths which have more contributions to the image. In the following sections, the algorithm of MMLT will be introduced step by step.

4.1 Initilisation of the Average Image Contributions

At the beginning, MMLT has to specifically choose a path length k to decide the sub-paths' length. Subsequently, bidirectional path tracing can be used to generate a camera path and a light path with the path length t and s respectively so as to evaluate the average image contribution of every path length k. According to Hachisuka *et al.* (2014), the formula of the average image contribution of each path k is formulated as below:

$$b_k \approx \frac{1}{N_k} \sum_{i=1}^{N_k} \hat{w_t^*}(\bar{u}) \hat{C^*}(\bar{u}), \tag{4.3}$$

where $k \in$ [MinPathLength, MaxPathLength], and the min path length and the max path length can be decided by users to determine how many depths rays would trace. These average image contributions can be considered as the probability of choosing this path length k for each sample in rendering.

4.2 The Mutation Strategies

The mutation strategies presented by Kelemen *et al.* (2002) were used in MMLT. For each sample, a random number is used to compare with the probability of the large step to determine which mutation method would be adopted. For example, if the random number is greater than the probability of the large step, it will use a mutation function to do the local mutation. Otherwise, it will randomly generate a new sample for the global mutation. In their strategy, samples can be created separately, and the implementation of this strategy is easier to implement compared to Veach and Guibas (1997). In MMLT, the new sample is called the proposal path, and the old sample is called the current path.

4.3 The Path Contributions to the Image

After choosing the mutation method, the contribution of the proposal path is calculated using the method of bidirectional path tracing for generating paths and then computes the acceptance probability a according to Equation 4.2. Simply, the value of the acceptance probability is the contribution luminosity of the proposal path divided by the contribution luminosity of the current path. Finally, the contributions of the proposal path and the current path are scaled using the equations from Kelemen *et al.* (2002) and then accumulates the contributions to the corresponding pixels. The scale equation for the current path is shown as below:

$$scale = \frac{(1-a)\cdot(k+2)}{(I^*(\mathbf{u})/b_k + P_{large})\cdot p_k(\bar{x})}, \tag{4.4}$$

where $I^*(\mathbf{u})$ is the scalar contribution function, P_{large} is the probability of the large step, and $p_k(\bar{x})$ here is the probability contribution function of the total path length k. Additionally, the scale equation for the

proposal path is:

$$scale = \frac{(a + LargeStep) \cdot (k + 2)}{(I^*(\mathbf{u})/b_k + P_{large}) \cdot p_k(\bar{x})}. \tag{4.5}$$

Here $LargeStep$ is 0 if the mutation is not a large step and 1 otherwise.

4.4 The Pseudo-Code of MMLT

The pseudo-code of this algorithm is shown as below:

---

initialise the average image contribution of each path length k
calculate Current[MaxPathLength] for each path length k
for each sample
  randomly choose a path length k
  if a random number > the probability of the large step
    assign new random numbers to Proposal
  else
    Proposal = mutate Current[k]
  use the first random number of Proposal to obtain t and s
  generate a camera path with the length t
  generate a light path with the length s
  connect the camera path with the light path
  Proposal.contribution = the contribution of the combined path
  calculate the acceptance probability a
  accumulate the path contributions to the corresponding pixels
  if a random number <= the acceptance probability
    Current[k] = Proposal

---

where Current and Proposal are a Markov chain data structure.

Chapter 5

Implementation

C++ was chosen as the programming environment, because it has many useful standard libraries to efficiently make a functional program. Figure 5.1 shows the class design of the rendering engine. It should be mentioned that most classes were finished for former work. This project mainly carried out bidirectional path tracing and multiplexed Metropolis light transport, though some classes were needed to modify slightly for this project.

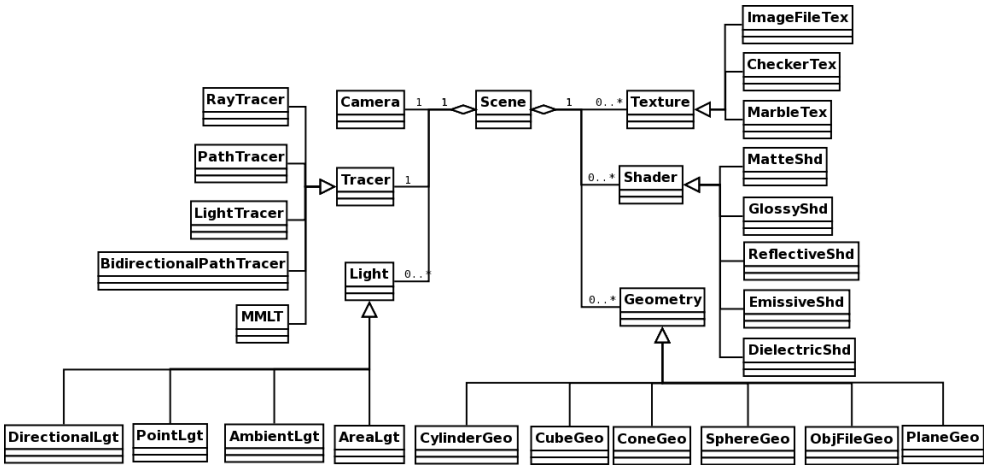


Figure 5.1: *The basic class design of the rendering engine.*

5.1 Previous Work

This project was extended from the rendering engine which was written for former assignments. The engine was implemented from scratch for the first assignment of the course, and the goal of this assignment needed to complete a ray tracer. During this period, most geometries, non-area lights, basic materials and textures were completed. Furthermore, the bounding volume hierarchy was also carried out for the acceleration structure to increase the performance.

Following this, path tracing was chosen for the assignment of the next major unit. The bidirectional scattering distribution function (BSDF) models such as matte, glossy, reflective and dielectric materials were finished while doing path tracing, and thus other global illumination rendering algorithms could apply these materials for rendering. Another thing to mention is that depth of field and HDRI lighting were also implemented, but they are only supported for path tracing in the rendering engine.

Subsequently, although it was not essential to carry out any programs for the next unit's assignment, light tracing was finished due to the fact that it was beneficial for future research, especially for rendering topics. As a result, before starting working the master project, path tracing and light tracing have already been achieved, and they could be adopted to accomplish bidirectional path tracing which would be used in multiplexed Metropolis light transport.

5.2 Implementation of Bidirectional Path Tracing

In order to achieve multiplexed Metropolis light transport, bidirectional path tracing should be completed first so as to utilise its method to construct paths. In terms of bidirectional path tracing, the way to generate camera pahts and light paths is very similar to path tracing and light

tracing, but it needs to connect each vertex of the camera path with each vertex of the light path if there is no obstacle between them. After connecting vertices, multiple importance sampling can be used to estimate the contribution of each connected path. The BidirectioanlPathTracer class design in the implementation is shown in Figure 5.2.

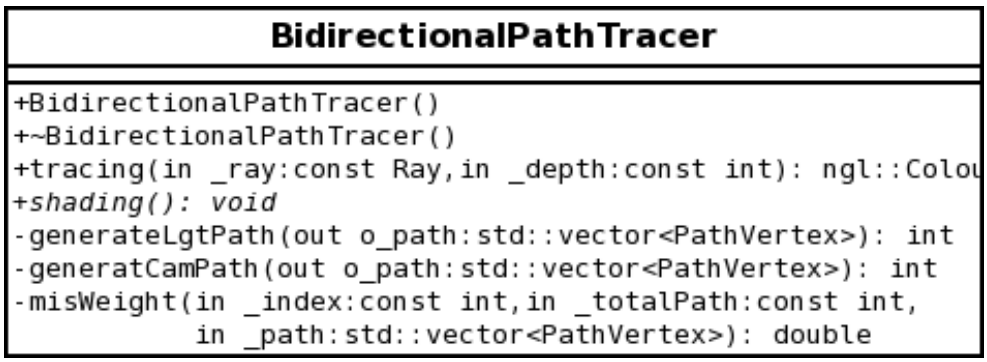


Figure 5.2: *The BidirectionalPathTracer class design.*

During the period of the implementation, the biggest difficulty is to correctly calculate the weighted value of the contribution of each connected path. Totally, around one month was spent on implementing and testing bidirectional path tracing.

5.3 Implementation of Multiplexed Metropolis Light Transport

It is undoubted that the final goal of the master project was to carry out multiplexed Metropolis light transport. First of all, it should be noted that Hachisuka *et al.* (2014) provided a simplified implementation of this rendering algorithm in one file which could be obtained from their website, and it is very helpful to more efficiently comprehend how their algorithm works. Figure 5.3 shows the MMLT class design in the implementation.

The function ″pathProbability″ was used to estimate the probability density of the connected path, and the function ″mutate″ is the one used in Kelemen *et al.* (2002).

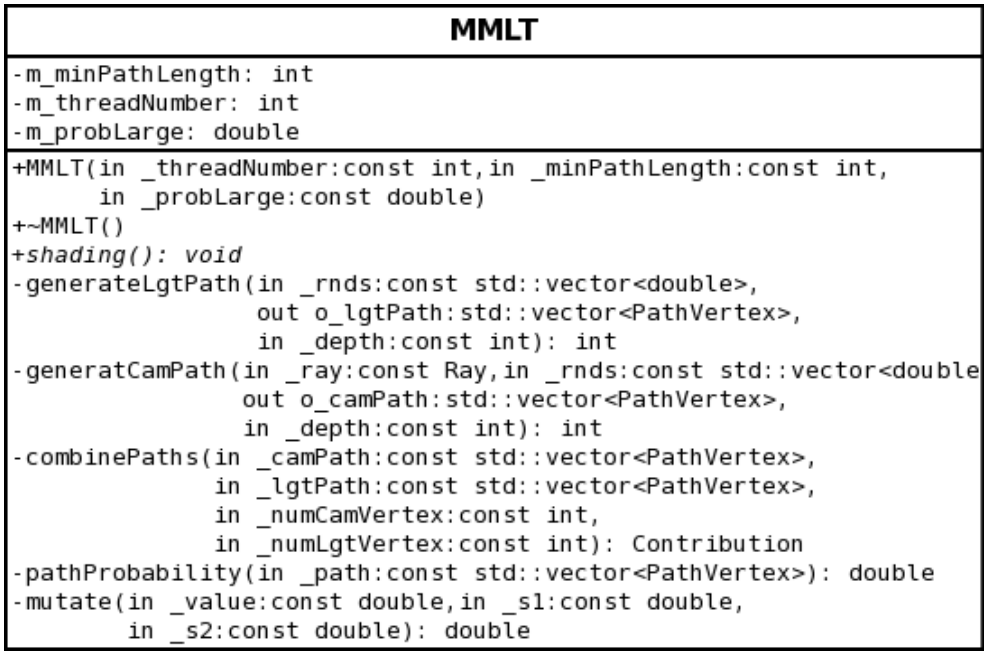


Figure 5.3: *The MMLT class design.*

5.4 The Graphical User Interface

It was simple to utilise Qt to make a graphical user interface and OpenGL to do pre-visualisation. The initial program framework was modified from Macey (2014), and the interface is shown in Figure 5.4.

The interface supports path tracing, light tracing, bidirectional path tracing and multiplexed Metropolis light transport. A geometry shader was adopted to make a resolution gate, and thus users can set the parameters of the image width and height to see the region which will be rendered. Furthermore, users can move the mouse to control the position and direction of the camera, and the camera information is shown at the top-left corner of the interface.

Boost Thread was used for starting rendering, and thus the engine can execute the rendering program in the background. To be convenient, a XML parser was written for loading and saving the scene file, and the scene description using XML can be seen in Appendix A.

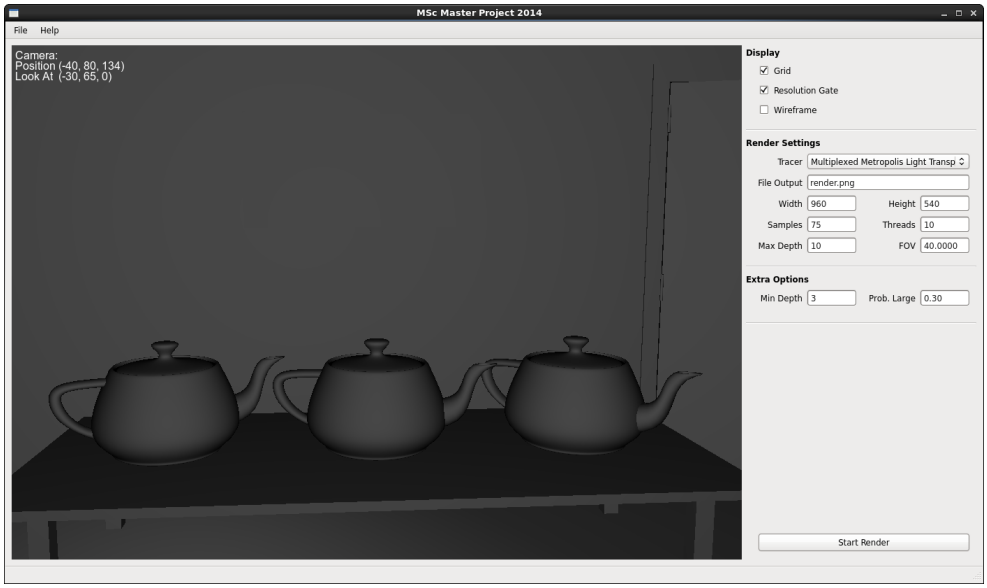


Figure 5.4: *The graphical user interface.*

5.5 Other Applications

5.5.1 Multi-Threading

It was not difficult to achieve multi-threading rendering using OpenMP. All global illumination rendering algorithms implemented in the rendering engine are supported for multi-threading. As a result, the computation time can be reduced dramatically.

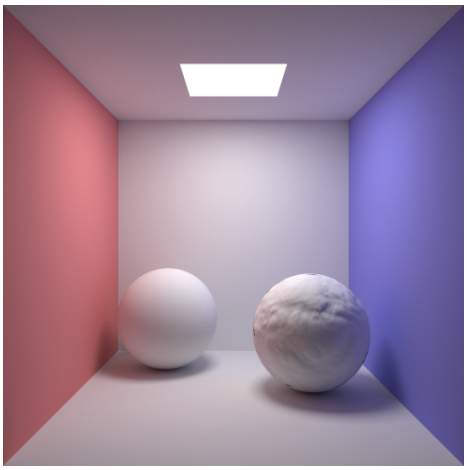
5.5.2 Render View

After finishing rendering, the program will create a window for viewing the rendered image. SDL was adopted to simply make this render view window, and its basic program framework was modified from Macey (2014).

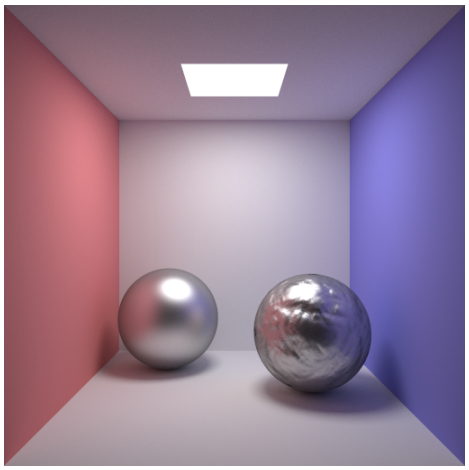
5.5.3 Bump Mapping

Caha and Peroutkova provided a simplified implemention of bump mapping on their website. The basic idea was to slightly modify the surface

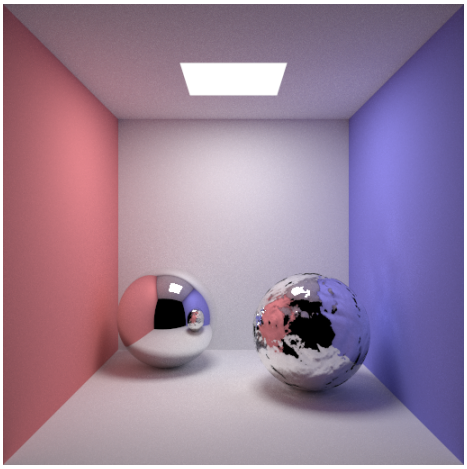
normal according to the luminance of the texture colour. Figure 5.5 shows an example with and without bump mapping rendered by the rendering engine.



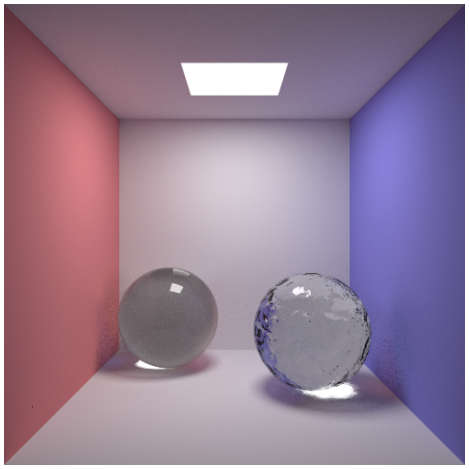
(a) Matte Material



(b) Glossy Material



(c) Reflective Material



(d) Dielectric Material

Figure 5.5: *(a) Matte Material. (b) Glossy Material. (c) Reflective Material. (d) Dielectric Material.*

Chapter 6

Results

In this chapter, it will show several results rendered by the rendering engine with various render settings. The hardware information is shown in Figure 6.1. The following rendering tests used a room which does not have any direct illumination in the field of view, so it could fully demonstrate the efficiency of multiplexed Metropolis light transport for solving indirect illumination problems.

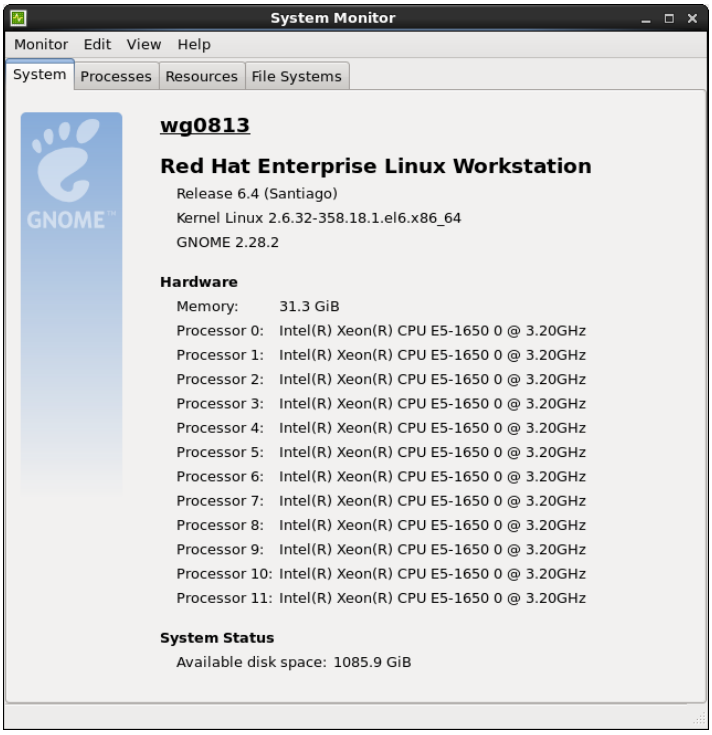
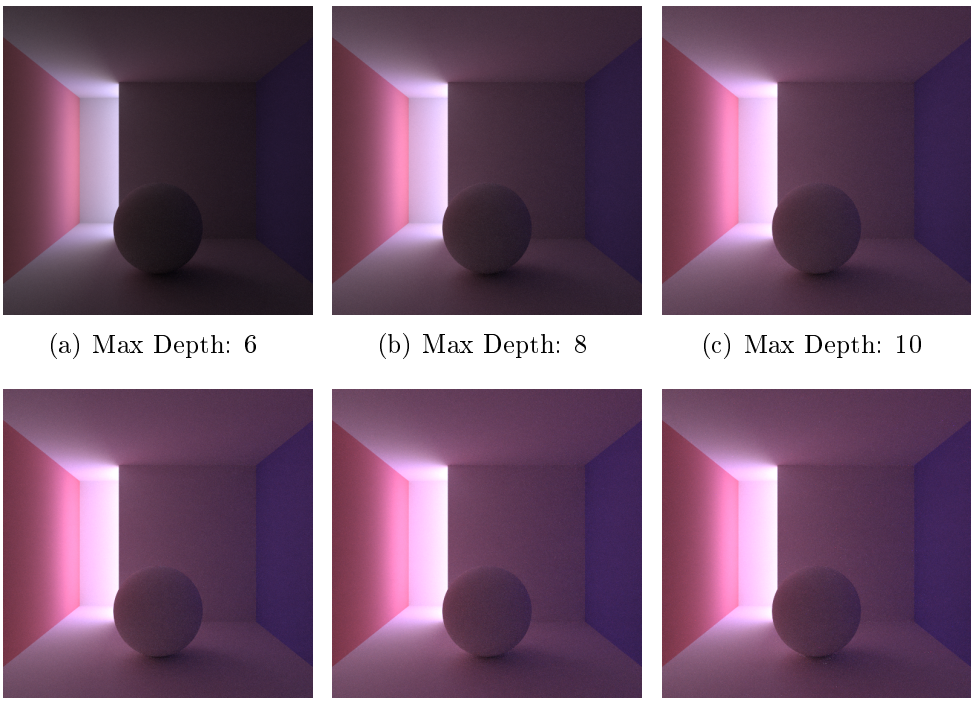


Figure 6.1: *The information of the computer for rendering tests.*

6.1 Different Depths for Tracing

The images shown in Figure 6.2 were rendered using different max path lengths to confirm that the convergence is working in the implementation.



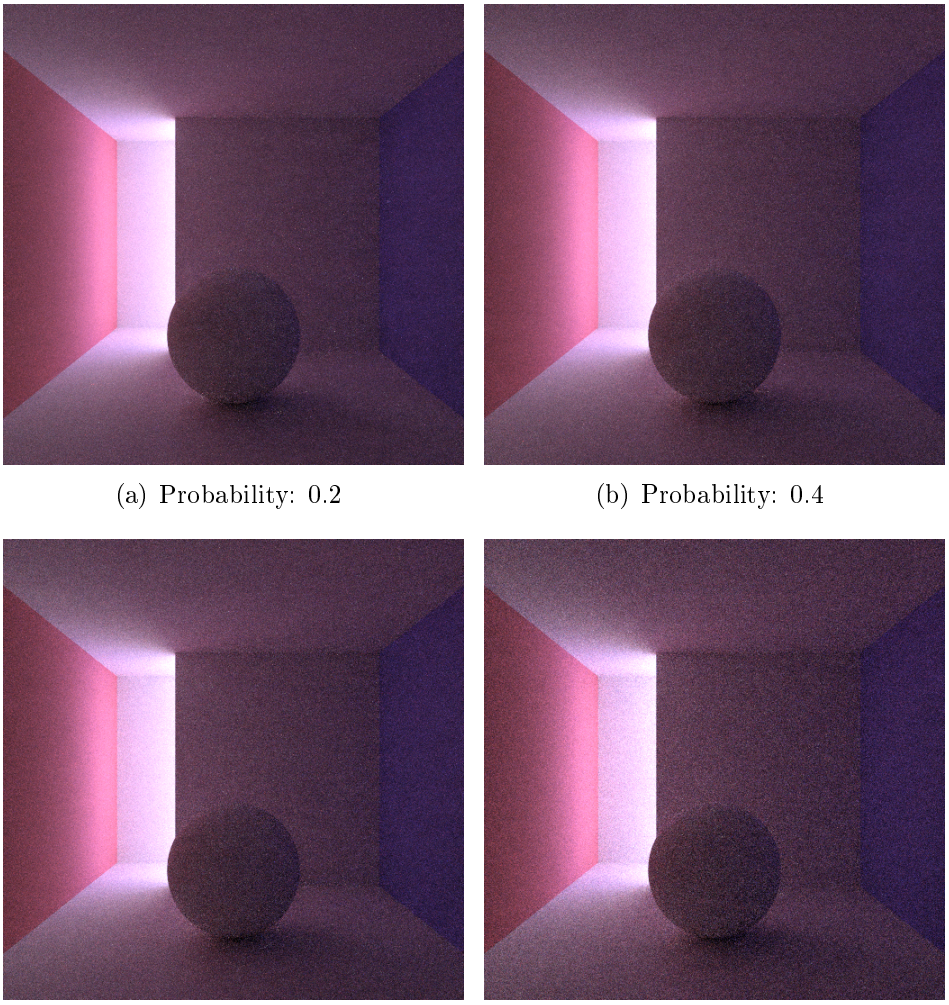
(a) Max Depth: 6 (b) Max Depth: 8 (c) Max Depth: 10

(d) Max Depth: 12 (e) Max Depth: 14 (f) Max Depth: 16

Figure 6.2: *These images were rendered using multiplexed Metropolis light transport. The min path length was 3, and the probability of the large step was 0.3. Furthermore, they used 2,304 sampes per pixel and 12 threads. (a) The computation time: 00:58:05. (b) The computation time: 01:12:25. (c) The computation time: 01:23:59. (d) The computation time: 01:36:41. (e) The computation time: 01:45:29. (f) The computation time: 01:58:24.*

6.2 Different Probabilities of the Large Step

In this section, it used various probabilities of the large step to see the difference. The image shown in Figure 6.3 were rendered using 256 sampes per pixel and 12 threads. The min path length and the max path length were 3 and 10 respectively.



(a) Probability: 0.2 (b) Probability: 0.4

(c) Probability: 0.6 (d) Probability: 0.8

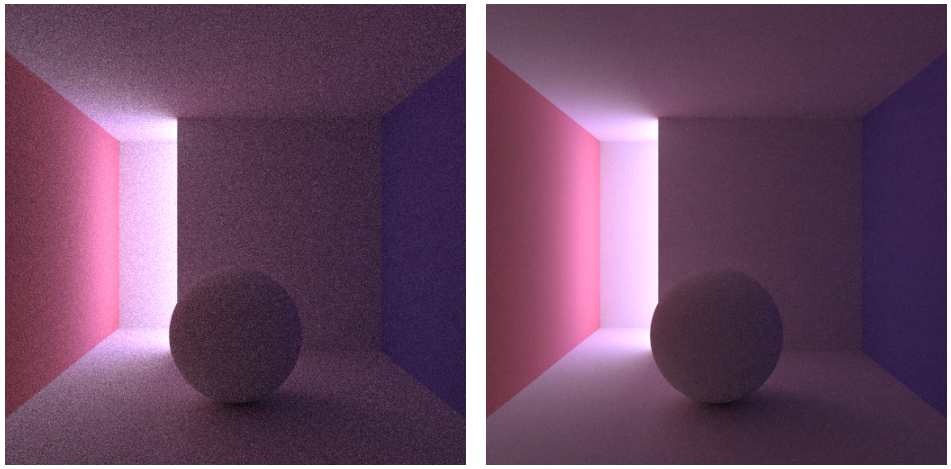
Figure 6.3: *(a) The computation time: 00:09:26. (b) The computation time: 00:08:58. (c) The computation time: 00:07:56. (d) The computation time: 00:07:16*

It demonstrates that using lower probability values would spend more computation time by reason that it needed to do more local mutation steps, but it could render better images compared to those using higher

probability values. Although the difference of the computation time between images is not too large, it would still affect the performance if using a large amount of samples and fewer threads.

6.3 Comparison

The images shown in Figure 6.4 took the same computation time using bidirectional path tracing and multiplexed Metropolis light transport respectively. The computation time was around 48 munites. Furthermore, the max path length was 10, and the number of the threads was 10. It can be seen that multiplexed Metropolis light transport could render better quality images than bidirectional path tracing.

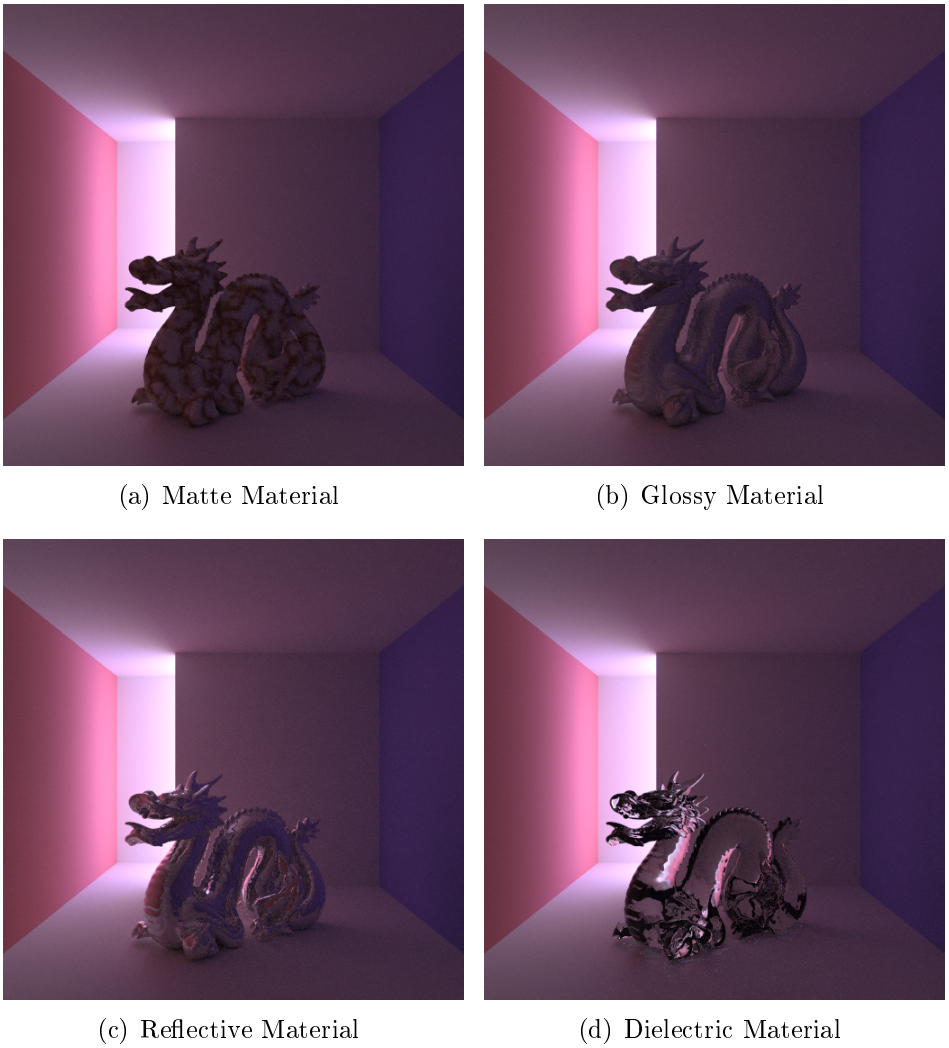


(a) Bidirectional Path Tracing (b) MMLT

Figure 6.4: *A comparison between bidirectional path tracing and multiplexed Metropolis light transport.*

6.4 Different BSDF Materials

In this section, it demonstrates the rendered images using different BSDF materials. The Stanford dragon model was taken from MrBluesummers (2010). In Figure 6.5 (a), the dragon was assigned a procedural marble texture which was modified from Shirley and Morley (2000). These images were rendered using 5,184 sample per pixel and 12 threads. The min path length and the max path length were 3 and 10 respectively. In addition, the probability of the large step was 0.3.



(a) Matte Material (b) Glossy Material

(c) Reflective Material (d) Dielectric Material

Figure 6.5: *These images were rendered using multiplexed Metropolis light transport. (a) The computation time: 38:24:17. (b) The computation time: 41:52:07. (c) The computation time: 45:37:14. (d) The computation time: 45:45:57.*

6.5 The Door Scene

This door scene first shown by Veach and Guibas (1997) is a well-known scene for the rendering test. Figure 6.6 shows the rendered result of the door scene using multiplexed Metropolis light transport. The models and textures were taken from Lehtinen *et al.* (2013). They remodeled this door scene's models which could be available from their website. Additionally, in order to reduce the rendering time, some models were replaced by the geometries provided in the rendering engine. It is undoubted that multiplexed Metropolis light transport can efficiently solve indirect illumination issues. However, in this project, there was not a proper anti-aliasing technique adopted in multiplexed Metropolis light transport. Hence, it is obious to see aliasing near the door.

Figure 6.6: *The door scene image. This image was rendered using 22,500 samples per pixel and 10 threads. The min path length and the max path length were 3 and 10 respectively, and the probability of the large step was 0.3. The computation time: 61:41:00.*

Chapter 7

Conclusion

Clearly, it is better to draw a comparison between different Metropolis light transport rendering algorithms. To implement other Metropolis light transport rendering algorithms would be a task for further research. In addition, more efficient acceleration structures such as bounding interval hierarchies could be considered in order to reduce the computation time. Using the GPU to create an interactive re-lighting tool might be another good direction for future work. To increase the image quality, a good anti-aliasing technique should be also considered in the implementation.

From Hachisuka *et al.* (2014), there are still some applications which are worth developing. For example, this algorithm is able to be integrated into the existing Markov chain Monte Carlo rendering algorithms because of its relatively straightforward implementation and satisfying efficiency. Moreover, it is also practical and simple to apply this algorithm to volume light transport.

In summary, multiplexed Metropolis light transport was implemented successfully and demonstrates the efficiency of this algorithm to dramatically solve indirect illumination problems. The rendered images are better than several global illumination rendering algorithms when the light sources are implicit. The parameters in the graphical user interface can be easily adjusted by users for various render settings. Although the

pre-visualisation is very basic, it is still helpful for users to conveniently browse a 3D scene in advance. Overall, the objective of this project proposed at the initial presentation was nearly achieved. With more time, thorough testing could be done to obtain further data for a more detailed comparison and analysis.

Bibliography

Adamsen M., 2009. Bidirectional path tracer. Available at `http://www.maw.dk/wp-content/uploads/2013/04/chart.jpg` [Accessed 25 August 2014].

Arvo J., 1986. Backward ray tracing. *In Developments in Ray Tracing, ACM SIGGRAPH Course Notes, 259-263.*

Caha M. and Peroutkova H. Bump mapping. Available at `http://rendering.aspone.cz/BumpMapCode.aspx` [Accessed 27 August 2014].

Dutre P., Bekaert P. and Bala K., 2006. *Advanced Global Illumination, Second Edition.* A K Peters, Ltd.

Geyer C. J. and Thompson E. A., 1995. Annealing markov chain monte carlo with applications to ancestral inference. *Journal of the American Statistical Association 90, 431, 909-920.*

Hachisuka T., Kaplanyan A. S. and Dachsbacher C., 2014. Multiplexed metropolis light transport. *ACM Transactions on Graphics (TOG).*

Kajiya J. T., 1986. The rendering equation. *Computer Graphics (Proceedings of SIGGRAPH '86), 143-150.*

Kelemen C., Szirmay-Kalos L., Antal G. and Csonka F., 2002. A simple and robust mutation strategy for the metropolis light transport algorithm. *Computer Graphics Forum.*

Lafortune E. P. and Willems Y. D., 1993. Bi-directional path tracing. *In Compugraphics '93, 145-153.*

Lehtinen J., Karras T., Laine S., Aittala M., Durand F. and Aila T.,

2013. Gradient-domain metropolis light transport. *ACM Transactions on Graphics (Proc. SIGGRAPH) 32, 4.*

Macey J., 2014. Jon macey computer animation pages. Available at `http://nccastaff.bournemouth.ac.uk/jmacey/GraphicsLib/Demos/index.html` [Accessed 27 August 2014].

Marinari E. and Parisi G., 1992. Simulated tempering: a new monte carlo scheme. *EPL (Europhysics Letters) 19, 6, 451.*

Metropolis N., Rosenbluth A. W., Rosenbluth M. N., Teller A. H. and Teller E., 1953. Equation of state calculation by fast computing machines. *The Journal of Chemical Physics 21, 1087.*

MrBluesummers , 2010. Stanford dragon model. Available at `http://www.mrbluesummers.com/3572/downloads/stanford-dragon-model` [Accessed 25 August 2014].

Shirley P. and Morley R. K., 2000. *Realistic Ray Tracing, Second Edition.* A K Peters, Ltd.

Suffern K., 2007. *Ray Tracing from the Ground Up.* A K Peters, Ltd.

Veach E. *Robust Monte Carlo Methods for light transport simulation.* PhD thesis, PhD these, Stanford University, USA, 1998.

Veach E. and Guibas L., 1997. Metropolis light transport. *In SIGGRAPH '97, 65-76.*

Veach E. and Guibas L. J., 1995. Optimally combinning sampling techniques for monte carlo rendering. *In SIGGRAPH '95, 419-428.*

Appendix A

Scene Description

```
<Scene>
 <Settings>
  <Width value="512"/> <!– the image width –>
  <Height value="512"/> <!– the image height –>
  <Sample value="4"/> <!– N * N samples –>
  <Thread value="12"/> <!– the number of threads –>
  <Tracer name="MMLT"/> <!– Tracer: MMLT, BPT, LT, PT –>
  <MaxPathLength value="10"/> <!– the max path length –>
  <MinPathLength value="3"/> <!– the min path length –>
  <ProbLarge value="0.3"/> <!– the probability of the large step –>
  <Output name="'render.png'/> <!– the output file name –>
  <EnvMap name="hdri.exr"/> <!– the environment map file –>
 </Settings>
 <Geometry Type="Plane" ID="0"> <!– a plane geometry –>
   <Position x="4" y="0" z="0"/> <!– the position –>
  <Rotation x="0" y="0" z="90"/> <!– the rotation –>
  <Width value="8"/> <!– the width of the plane –>
  <Depth value="8"/> <!– the height of the plane –>
  <Shader value="2"/> <!– the shaders ID –>
 </Geometry>
 <Geometry Type="Sphere" ID="1"> <!– a sphere geometry –>
  <Position x="0" y="-2.5" z="0"/> <!– the position –>
```

```
  <Rotation x="0" y="0" z="0"/> <!– the rotation –>
  <Radius value="1.5"/> <!– the radius –>
  <Shader value="3"/> <!– the shaders ID –>
 </Geometry>
 <Geometry Type="ObjFile" ID="2"> <!– an obj file geometry –>
  <Position x="0" y="-4" z="0"/> <!– the position –>
  <Rotation x="0" y="0" z="0"/> <!– the rotation –>
  <File name="dragon.obj"/> <!– the obj file name –>
  <SmoothShading value="1"/> <!– on : 1, off : 0 –>
  <Shader value="0"/> <!– the shaders ID –>
 </Geometry>
 <Light Type="Area" ID="0"> <!– an square area light –>
  <Position x="0" y="3.9999" z="0"/> <!– the position –>
  <Rotation x="0" y="0" z="180"/> <!– the rotation –>
  <Width value="2.5"/> <!– the square's width –>
  <Depth value="2.5"/> <!– the square's depth –>
  <Intensity value="14"/> <!– the intensity –>
  <Colour r="1" g="1" b="1" t="0"/> <!– the light colour –>
 </Light>
 <Light Type="Area" ID="0"> <!– an square area light –>
  <Position x="0" y="3.9999" z="0"/> <!– the position –>
  <Rotation x="0" y="0" z="180"/> <!– the rotation –>
  <Radius value="4"/> <!– the sphere's radius –>
  <Intensity value="14"/> <!– the intensity –>
  <Colour r="1" g="1" b="1" t="0"/> <!– the light colour –>
 </Light>
 <Shader Type="Matte" ID="0"> <!– a matte shader –>
  <Diffuse r="0.9" g="0.9" b="0.9" t="0"/> <!– the diffuse colour
–>
  <Ambient r="0" g="0" b="0" t="0"/> <!– the ambient colour –>
  <Bump value="0"/> <!– the texture ID –>
 </Shader>
 <Shader Type="Reflective" ID="1"> <!– a reflective shader –>
  <Reflect r="1" g="1" b="1" t="0"/> <!– the reflection colour –>
```

```
  <Bump value="0"/> <!– the texture ID –>
 </Shader>
 <Shader Type="Glossy" ID="2"> <!– a glossy shader –>
  <Diffuse r="0.15" g="0.15" b="0.15" t="0"/> <!– the diffuse
colour –>
  <Ambient r="0" g="0" b="0" t="0"/> <!– the ambient colour –>
  <Specular r="0.85" g="0.85" b="0.85" t="0"/> <!– the specular
colour –>
  <Shininess value="10"/> <!– the shininess –>
  <Bump value="0"/> <!– the texture ID –>
 </Shader>
 <Shader Type="Dielectric" ID="3"> <!– a dielectric shader –>
  <Reflect r="1" g="1" b="1" t="0"/> <!– the reflection colour –>
  <Refract r="1" g="1" b="1" t="0"/> <!– the refraction colour –>
  <iIOR value="1.5"/> <!– the incident index of refraction –>
  <oIOR value="1.5"/> <!– the exiting index of refraction –>
  <Bump value="0"/> <!– the texture ID –>
 </Shader>
 <Camera Type="Camera" ID="0"> <!– a camera object –>
  <Position x="0" y="0" z="-12"/> <!– the position –>
  <LookAt x="0" y="0" z="0"/> <!– look at the position –>
  <FOV value="53.1301"/> <!– field of view –>
  <DOF value="0"/> <!– depth of view, on : 1, off : 0 –>
  <F-Length value="12"/> <!– the focal length –>
  <F-Stop value="25"/> <!– the f-stop value –>
 </Camera>
 <Texture Type="Checker" ID="1"> <!– a checker texture –>
  <Colour1 r="0.5" g="0.5" b="0.5" t="0"/> <!– the first colour –>
  <Colour2 r="1" g="1" b="1" t="0"/> <!– the second colour –>
  <Repeat u="1" v="1"/> <!– the uv coordinate –>
 </Texture>
 <Texture Type="ImageFile" ID="2"> <!– an image file texture –>
  <File name="fileName.png"/> <!– the file name –>
  <Repeat u="1" v="1"/> <!– the uv coordinate –>
```

```
 </Texture>
 <Texture Type="Marble" ID="3"> <!– a 3D marble texture –>
   <Colour1 r="0.8" g="0.8" b="0.8" t="0"/> <!– the first colour –>
   <Colour2 r="0.4" g="0.2" b="0.1" t="0"/> <!– the second colour
–>
   <Colour3 r="0.06" g="0.04" b="0.02" t="0"/> <!– the third
colour –>
 </Texture>
</Scene>
```