# Na tomto místě bude oficiální zadání vaší práce

- Toto zadání je podepsané děkanem a vedoucím katedry,
- musíte si ho vyzvednout na studiijním oddělení Katedry počítačů na Karlově náměstí,
- v jedné odevzdané práci bude originál tohoto zadání (originál zůstává po obhajobě na katedře),
- ve druhé bude na stejném místě neověřená kopie tohoto dokumentu (tato se vám vrátí po obhajobě).

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Graphics and Interaction

Master's Thesis

# Efficient methods for computing complex global illumination effects

*Bc. Robin Hub*

Supervisor: Ing. Bittner Jiri, Ph.D.

Study Programme: Open Informatics

Field of Study: Computer Graphics and Interaction

December 19, 2013

# Aknowledgements

I would like to thank Ing. Jiří Bittner, Ph.D. for his professional supervision and advices. Doc. Ing. Vlastimil Havran, Ph.D. for his valuable comments and remarks. Martin Štýs for proofreading the theoretical part. Iliyan Georgiev for allowing me to use his scenes for the evaluation of the implementation.

# Declaration

I hereby declare that I have completed this thesis independently and that I have listed all the literature and publications used.
I have no objection to usage of this work in compliance with the act §60 Zákon č. 121/2000Sb. (copyright law), and with the rights connected with the copyright act including the changes in the act.

In Prague on December 19, 2013 ...........................................................

# Abstract

Computing a photo realistic image of an arbitrary virtual scene in an affordable amount of time is still an open problem. Often, one has to trade certain illumination effects for speed. However, the recently proposed global illumination methods show that the solution to the problem does not only lie in increasing the speed of the computer hardware, but also in careful algorithm design.

This thesis presents a comprehensive overview of the theory of the global illumination, discusses the fundamental rendering algorithms using the presented concepts and a recent method, vertex connection and merging (VCM)[GKDS12], is presented in detail as a combination of the basic algorithms. The text follows with the description of the architecture of an advanced photorealistic rendering software and with the description of the VCM implementation within an existing rendering toolkit. Then we evaluate the VCM algorithm, compare it to the other algorithms, and present how its input parameters affect the resulting image.

# Contents


**1 Introduction** **1**

1.1 Global Illumination . . . 1

1.2 Approaches to Global Illumination . . . 2

**2 Preliminaries** **3**

2.1 Foundations . . . 3

2.1.1 Common definitions and notations . . . 3

2.1.2 Abstract model of light . . . 7

2.1.3 Radiometry and radiometric quantities . . . 7

2.1.4 Light surface interaction . . . 9

2.1.4.1 BRDF . . . 10

2.1.4.2 BTDF . . . 11

2.1.4.3 BSDF . . . 12

2.1.5 Light transport and measurement . . . 13

2.1.6 Monte Carlo integration . . . 15

2.1.6.1 Introduction . . . 16

2.1.6.2 Importance sampling . . . 17

2.1.6.3 Multiple importance sampling . . . 19

2.1.6.4 Russian roulette . . . 20

2.2 Path sampling methods . . . 20

2.2.1 Path space formulation of light transport . . . 20

2.2.2 Path tracing . . . 23

2.2.3 Bidirectional path tracing . . . 26

2.3 Density estimation methods . . . 29

2.3.1 A short introduction to particle tracing . . . 29

2.3.2 Photon mapping . . . 29

2.3.2.1 Photon map generation . . . 30

2.3.2.2 Rendering using photon map . . . 31

2.3.2.3 Extensions . . . 32

2.3.3 Progressive photon mapping . . . 32

2.3.4 Stochastic progressive photon mapping . . . 35

2.3.5 Bidirectional photon mapping . . . 35

**3 Vertex connection and merging** **37**
3.1 Vertex connection . . . 38
3.2 Vertex merging . . . 38
3.3 Measurement estimation . . . 39
3.4 The VCM algorithm . . . 40
3.5 Convergence . . . 42

**4 Implementation** **43**
4.1 Global Illumination Rendering Toolkit (GIRT) . . . 43
4.1.1 Design goals . . . 44
4.1.2 Library overview . . . 45
4.1.2.1 Scene . . . 45
4.1.2.2 Materials . . . 48
4.1.2.3 Camera models . . . 49
4.1.2.4 Rendering . . . 49
4.1.2.5 Image filtering and tone mapping . . . 51
4.1.2.6 Implementation details . . . 51
4.2 Vertex connection and merging implementation . . . 53

**5 Results** **61**
5.1 Testing correctness . . . 61
5.2 Rendering algorithms comparison . . . 63
5.3 Optimizing VCM input parameters . . . 65
5.4 Increasing the number of vertex merging samples . . . 66
5.5 Computing image for more merging radii at once . . . 67

**6 Conclusion** **75**
6.1 Future work . . . 75

**Bibliography** **77**

**A Materials definition file example** **81**

**B Lights definition file example** **83**

**C Benchmark definition file example** **85**

**D Image gallery** **87**

**E Installation and user manual** **91**
E.1 Requirements . . . 91
E.2 GIRay application . . . 91
E.3 GIRay Benchmark application . . . 93
E.4 Building from source code . . . 93

**F DVD Content** **95**

# List of Figures


1.1 The bathroom scene created for the purposes of this thesis and rendered by our implementation of the VCM algorithm. . . . . . . . . . . . . . . . . . . . 2

2.1 A direction $\overrightarrow{\omega}$ and the associated solid angle $d\sigma(\overrightarrow{\omega})$. . . . . . . . . . . . . . 4
2.2 A direction $\overrightarrow{\omega}$ and the associated projected solid angle $d\sigma_{\mathbf{x}}^{\perp}(\overrightarrow{\omega})$. . . . . . . . 5
2.3 Geometric settings of two points . . . . . . . . . . . . . . . . . . . . . . . . 6
2.4 Incident radiance from the direction $\overrightarrow{\omega_i}$ is scattered uniformly to all directions. 10
2.5 Portion of the incident radiance from the direction $\overrightarrow{\omega_i}$ is scattered in the perfect reflection direction only. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
2.6 Portion of the incident radiance from the direction $\overrightarrow{\omega_i}$ is scattered in the perfect refraction direction only. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
2.7 The canonical uniform random variable. . . . . . . . . . . . . . . . . . . . . 17
2.8 Visualization of the inversion method. The random variable $\xi_1$ is mapped to the random variable $y \sim p_y$ through $P_y^{-1}$. . . . . . . . . . . . . . . . . . . . 18
2.9 A path of length $k = 4$. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21
2.10 A single measurement during unidirectional path tracing. . . . . . . . . . . 25
2.11 A single measurement during the path tracing with next event estimation without explicit connection to the camera vertex. . . . . . . . . . . . . . . . . . 26
2.12 A single measurement during the bidirectional path tracing. A light path with 2 vertices is connected to an eye path with 3 vertices. Dashed lines depict the explicit subpath connections. . . . . . . . . . . . . . . . . . . . . . . . . . . 27
2.13 A single traced photon path during the process of photon map generation. The yellow line depicts the light source, which is the start point of every photon path. Individual photon records created by tracing depicted photon path are shown as yellow disks. . . . . . . . . . . . . . . . . . . . . . . . . . . 30
2.14 Rendering from the photon map. Figure depicts the process of radiance estimation for two pixels. Red discs represent hitpoints created by tracing primary rays from the camera. Yellow discs represent photons stored in the photon map. A single thin arrow points in the direction from which is a particular photon incident. The circles visualizes the photon search radius. Note that we required 5 photons for a single radiance estimate. . . . . . . . . . . . . . . 32
2.15 Visualization of radiance estimate for a single pixel within bidirectional photon mapping. An eye path is traced through the scene. Photon map radiance estimate is performed at each eye vertex (red disc) and the results are combined using the multiple importance sampling . . . . . . . . . . . . . . . . . . 35

3.1 A path of length $k = 4$ sampled by the vertex connection technique $s = 2, t = 3$. 38
3.2 A path of length $k = 4$ sampled by the vertex merging technique $s = 3, t = 3$. 38
3.3 This figure depicts the vertex connection and merging estimate for a single pixel. Each eye path vertex (red disc) is connected to each light path vertex (yellow disc) as in bidirectional path tracing. Additionally, photon mapping radiance estimate is performed at each eye path vertex. . . . . . . . . . . . . . . 42

4.1 The architecture around the class **SceneCache**. . . . . . . . . . . . . . . . . . 46
4.2 The architecture around the class **Light**. . . . . . . . . . . . . . . . . . . . . . 47
4.3 The architecture around the class **Material**. . . . . . . . . . . . . . . . . . . . 48
4.4 The architecture around the class **Camera**. . . . . . . . . . . . . . . . . . . . 50
4.5 The architecture around the class **AbstractRenderer**. . . . . . . . . . . . . 50

D.1 The virtual model of the hallway at Department of Computer Graphics at Czech Technical University in Prague create by Tomáš Kraus within his Bachelor's thesis. Rendering took 3 hours. . . . . . . . . . . . . . . . . . . . . . . . . . 88
D.2 The car model created by Alex Kuntz and modified by Iliyan Georgiev. The scene is lit by a HDR environment map. Render time was 1 hour. . . . . . . . 88
D.3 An image of the living room scene, which was modeled by Iliyan Georgiev. Render time was 2 hours. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 89
D.4 Bathroom scene lit by a special light source that imitates sunny daylight through a window. Rendering took 1 hour. . . . . . . . . . . . . . . . . . . . . . 89
D.5 Mirror balls scene by Toshiya Hachisuka rendered in 1 hour. . . . . . . . . . . 90

E.1 A screenshot of the GIRay application. . . . . . . . . . . . . . . . . . . . . . . . 92
E.2 A screenshot of the GIRay application. . . . . . . . . . . . . . . . . . . . . . . . 93

# List of Tables

2.1 Common notations used in this thesis. . . . . . . . . . . . . . . . . . . . . . . . 4

5.1 The top left image shows the reference image produced by our implementation of path tracing with the next event estimation in 30 minutes. The top right image shows the same scene rendered by our implementation of vertex connection and merging in the same amount of time. The bottom left image visualizes the color difference between top images. The RMS difference is 0.006. The plot visualizes how the RMS difference from the reference evolved in time for both algorithms. The contour of the light source on the difference image is caused by the random sampling of pixel locations. See that the path tracer converges faster in this simple case (diffuse surfaces) than the VCM, since within 30 minutes it performed 3657 iterations while the VCM performed *only* 929 iterations. . . . . . . . . . . . . . . . . . . . . . . . . . . . 62

5.2 Tone mapped reference images of the scenes used for the evaluation of our VCM implementation. **Left:** *Mirror balls* scene by Toshiya Hachisuka **Middle:** *Living room* scene by Iliyan Georgiev **Right:** custom *Bathroom* scene . . 63

5.3 This table shows some profiling information of the last VCM iteration that was performed when the images for the tables 5.6, 5.7, and 5.8 were created. The table 5.4 provides column descriptions. . . . . . . . . . . . . . . . . . . . . 65

5.4 This table provides column descriptions of the table 5.3. All times are in milliseconds and bound to the last iteration. . . . . . . . . . . . . . . . . . . . . 65

5.5 The result of VCM rendering after 1 minute and after 1 hour of the three scenes used for the evaluation of our VCM implementation, Mirror balls, Living room, and Bathroom. First column shows the complete image while the middle column shows its vertex merging contribution and the right column shows its vertex connection contribution. The number of iterations accomplished by the algorithm so far is also provided. . . . . . . . . . . . . . . . . . . . . . . . . 68

5.6 The result of rendering the *Mirror balls* scene after 60 minutes by vertex connection and merging, bidirectional path tracing, bidirectional photon mapping, and path tracing with the next event estimation. Number of iterations (samples per pixel) is also provided. The bottom left image shows the reference image computed by the VCM algorithm in 5 hours. The plot in the bottom right corner shows the evolution of the RMS difference from the reference image for individual methods. The spikes in convergence of the VCM and the BPM were caused by the current merging radius and photons densities together at a particular time. . . . . . 69
5.7 The result of rendering the *Living room* scene after 60 minutes by vertex connection and merging, bidirectional path tracing, bidirectional photon mapping, and path tracing with the next event estimation. Number of iterations (samples per pixel) is also provided. The bottom left image shows the reference image computed by the VCM algorithm in 5 hours. The plot in the bottom right corner shows the evolution of the RMS difference from the reference image for individual methods. . . . . . 70
5.8 The result of rendering the *Bathroom* scene after 60 minutes by vertex connection and merging, bidirectional path tracing, bidirectional photon mapping, and path tracing with the next event estimation. Number of iterations (samples per pixel) is also provided. The bottom left image shows the reference image computed by the VCM algorithm in 5 hours. The plot in the bottom right corner shows the evolution of the RMS difference from the reference image for individual methods. . . . . . 71
5.9 Results presented in this table show that sometimes it might be worth to increase the impact of vertex merging by increasing the number of light paths, which most likely increases the amount of photon records. All images (1024x768) on this table were created by the VCM algorithm after 60 minutes. The top left image was created with the merging radius 12.5 mm and every iteration the number of pixels light paths were traced. The next image to the right was created with the same initial merging radius but the number of light paths were quadrupled. To generate the bottom left image we doubled the merging radius and changed the number of light paths back to the number of pixels. The plot shows the RMS difference from reference of all three images in time and we can see that increased number of photons led to a better result. See the section 5.4 for more detailed discussion. . . . . . 72
5.10 The results of the vertex connection and merging for a various initial merging radii in time in the context of the *Living room* scene. The vertex merging at the second eye path vertex was turned on. The left column shows the result for the radius of 5mm, the middle column for 12mm, and the right column for 25mm. . . . . . 73
5.11 The results of the vertex connection and merging for a various initial merging radii in time in the context of the *Living room* scene. The vertex merging at the second eye path vertex was turned off. The left column shows the result for the radius of 5mm, the middle column for 12mm, and the right column for 25mm. . . . . . 74

# Chapter 1

# Introduction

Global illumination is the part of the computer graphics that deals with the photorealistic image synthesis. The quality of the result is the most important aspect. Of course, speed is also an important factor but not that important as it is in *real time* applications (e.g. games). Global illumination tries to provide digital images of virtual scenes which are indistinguishable from the photographs taken by today's cameras. This means global simulation of light transport without making any crucial simplification. The figure 1.1 shows an image rendered using the VCM global illumination algorithm presented in this thesis.

## 1.1 Global Illumination

The problem of global illumination is to find the amount of light incident on the camera sensor. This is a very short and vague definition but it should suffice for an intuitive understanding of the problem. Light emanates from its sources, travels through the environment and is absorbed or scattered upon interaction with the scene surface. In reality it takes some time before the light reaches a sensory surface. We can think about our virtual scene as about a closed system, in which the light emission is constant. That entails we are only interested in the incident amount of light on the camera sensor at any time at which the light distribution has already reached its equilibrium. In this case, we can assume that the light propagates at infinite speed, which means that time is not taken into account.

The thesis is restricted to the light model based on geometrical optics and radiometry. This choice allows the simulation of most global illumination effects (e.g. diffuse interreflection) as well as it keeps the computational cost relatively low. One of the very important effects not discussed here is participating media. Pharr et al.[PH10] offer a comprehensive discourse on this topic. Note that we could use the wave optics to model the light, which is relatively complicated but it allows us to simulate the effects of interference, diffraction, and polarization. Recall that our goal is to generate photo-realistic images, not exact physical measurements. The mentioned wave optics effects can be mostly omitted without anyone's notice.

Global illumination is a fragment in the process of computer generated image acquisition. Unfortunately it is not the only thing we have to solve in order to be able to present photo realistic pictures on today's low dynamic range (LDR) displays. The tone mapping

Figure 1.1: The bathroom scene created for the purposes of this thesis and rendered by our implementation of the VCM algorithm.

is the process that does exactly that enables to view high dynamic range images on LDR display. It takes a generated HDR image and produces its LDR version. Tone mapping is a very extensive topic on its own and is orthogonal to global illumination, therefore it is not discussed in this thesis. Reinhard et al.[RWP+10] or Banterle et al.[BADC11] offer a good insight to the topic.

## 1.2 Approaches to Global Illumination

Solving global illumination is in general a very computationally expensive task even if geometric optics is used and time is not taken into account. We can drop certain lighting effects like glossy inter-reflections, but this would decrease the realism of resulting image. These methods are not tackled in this text, because they can be derived from discussed methods.

Most methods that solves global illumination is stochastic and is based on estimating the result using Monte Carlo integration (e.g. path tracing). Each such method proposed till this day has its own weaknesses and strengths, which means that there is no omnipotent algorithm, that we could use. There are two main approaches to global illumination, one is based on sampling the space of light carrying paths (e.g. *Bidirectional path tracing*) and the other is using the density estimation (e.g. *Photon Mapping*). Georgiev et al.[GKDS12] proposed new method called *vertex connection and merging*(VCM), which is a practical combination of both mentioned approaches. VCM is a robust algorithm, which can handle a wide variety of scenes.

That said, we can start with more formal discussion of the theoretical background.

# Chapter 2

# Preliminaries

## 2.1 Foundations

The following text introduces the required theoretical background that is common to all global illumination algorithms discussed throughout this thesis. Theory is mostly presented in an informal and intuitive way to give the reader basic knowledge to understand later sections. References to the relevant resources are provided through the following text. Readers already familiar with the radiometry, light transport, and the Monte Carlo numerical integration can skip directly to the next section.

In the following text, we heavily use concepts from the measure theory [CK08][Bar95]. For example we will be working with measure spaces and Lebesgue integration. It is necessary that the reader understands the measure theory at least basically before continuing.

### 2.1.1 Common definitions and notations

The comprehensive overview of all common definitions and notations is given in the table 2.1. The following text provides a more elaborate introduction.

#### Surface

Scene surface $\mathcal{M}$ is a set of points $\mathbf{x} \in \mathbb{R}^3$. Every surface point has its normal denoted by $\mathbf{N_x}$. For practical reasons we restrict $\mathcal{M}$ to be a union of piecewise differentiable two-dimensional manifolds with boundaries (e.g. triangles).

We define the measure $A$ on $\mathcal{M}$ in a natural way, so that $A(D)$ is the surface area of a set $D \subseteq \mathcal{M}$. This way, given a function $f : \mathcal{M} \to \mathbb{R}$

$$\int_{\mathcal{M}} f(\mathbf{x})\, dA(\mathbf{x})$$

denotes the Lebesgue integral of $f$ with respect to the surface area measure.

| Notation | Description |
|---|---|
| $\mathcal{S}^2$ | The set of all directions, the unit sphere in $\mathbb{R}^3$ |
| $\mathcal{M}$ | The set of all surface points. |
| $\mathcal{H}^2_+(\mathbf{x})$ | Upward hemisphere at the point $\mathbf{x}$. |
| $\mathcal{H}^2_-(\mathbf{x})$ | Downward hemisphere at the point $\mathbf{x}$. |
| $T_{\mathcal{M}}(\mathbf{x})$ | Tangent space of point the point $\mathbf{x}$. |
| $V(\mathbf{x},\mathbf{y})$ | Visibility between points $\mathbf{x}$, $\mathbf{y}$. |
| $G(\mathbf{x},\mathbf{y})$ | Geometric term of points $\mathbf{x}$, $\mathbf{y}$. |
| $\langle f, g\rangle$ | Inner product of functions $f$, $g$. |
| $\delta_\sigma$ | Dirac's delta function with respect to the $\sigma$ measure. |
| $\mathbf{N_x}$ | Normal vector of the point $\mathbf{x}$. |
| $\mathbf{R_x}(\overrightarrow{\omega})$ | Reflection of the direction vector $\overrightarrow{\omega}$around the surface normal at the point $\mathbf{x}$. |
| $\mathbf{T_x}(\overrightarrow{\omega})$ | Direction of the transmitted light incident from $\overrightarrow{\omega}$at the point $\mathbf{x}$. |
| $\overrightarrow{\mathbf{xy}}$ | Unit length direction vector from $\mathbf{x}$ to $\mathbf{y}$ |
| $\cos\theta_{\mathbf{xy}}$ | $\mathbf{N_x}\cdot\overrightarrow{\mathbf{xy}}$ |

Table 2.1: Common notations used in this thesis.

**Directions and solid angle**

A direction $\overrightarrow{\omega}$ is represented by a unit vector in $\mathbb{R}^3$. The union of all directions in the mentioned space is denoted as $\mathcal{S}^2$, the unit sphere in $\mathbb{R}^3$. Measure $\sigma$ on $\mathcal{S}^2$ is defined in the way, such that $\sigma(D)$ represents the solid angle generated by a set of directions $D \subseteq \mathcal{S}^2$ (e.g. surface area of the unit sphere). As with the surface area measure, given a function $f : \mathcal{S}^2 \to \mathbb{R}$

$$\int_{D\subseteq\mathcal{S}^2} f(\overrightarrow{\omega})\, d\sigma(\overrightarrow{\omega})$$

denotes the Lebesgue integral of $f$ with respect to the solid angle $D$. Figure 2.1 shows a direction vector together with differential solid angle associated with it.

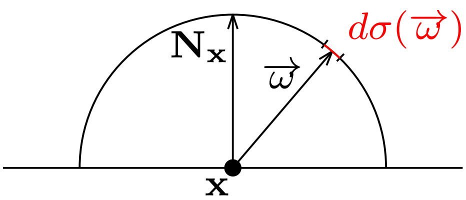


Figure 2.1: A direction $\overrightarrow{\omega}$ and the associated solid angle $d\sigma(\overrightarrow{\omega})$.

Given a surface point $\mathbf{x}$, we define the upper hemisphere

$$\mathcal{H}^2_+(\mathbf{x}) = \{\overrightarrow{\omega} \in \mathcal{S}^2 : \mathbf{N_x}\cdot\overrightarrow{\omega} > 0\},$$

the lower hemisphere

$$\mathcal{H}^2_-(\mathbf{x}) = \{\overrightarrow{\omega} \in \mathcal{S}^2 : \mathbf{N_x}\cdot\overrightarrow{\omega} < 0\}$$

and the tangent space

$$T_{\mathcal{M}}(\mathbf{x}) = \{\mathbf{y} \in \mathcal{M} : \mathbf{N_x} \cdot \overrightarrow{\mathbf{xy}} = 0\} \,.$$

### Projected solid angle

The next concept we will work with, is the projected solid angle. Given a surface point $\mathbf{x}$ and a set of directions $D \subseteq \mathcal{S}^2$, we define it as

$$\sigma^{\perp}_{\mathbf{x}}(D) = \int_D \mathbf{N_x} \cdot \overrightarrow{\omega}\, d\sigma(\overrightarrow{\omega})$$

This concept is for example very useful when it comes to computation of irradiance at particular surface point. Notice that $\sigma^{\perp}_{\mathbf{x}}(\mathcal{H}^2_+(\mathbf{x})) = \pi$, which is the area of the tangential unit disk at the point $\mathbf{x}$. Figure 2.2 shows the differential projected solid angle for a direction.

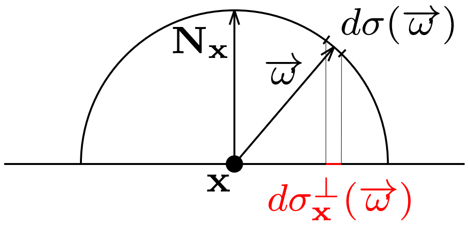


Figure 2.2: A direction $\overrightarrow{\omega}$ and the associated projected solid angle $d\sigma^{\perp}_{\mathbf{x}}(\overrightarrow{\omega})$.

### Geometric factor

Consider the geometric setting depicted by the figure 2.3. We define the geometric factor of two surface points $\mathbf{x}\,, \mathbf{y} \in \mathcal{M}$ as

$$G(\mathbf{x},\mathbf{y}) = \frac{|\mathbf{N_x} \cdot \overrightarrow{\mathbf{xy}}|\,|\mathbf{N_y} \cdot \overrightarrow{\mathbf{yx}}|}{\|\mathbf{x}-\mathbf{y}\|^2} V(\mathbf{x},\mathbf{y}) = \frac{|\cos\theta_{\mathbf{xy}}||\cos\theta_{\mathbf{yx}}|}{d^2} V(\mathbf{x},\mathbf{y})\,, \tag{2.1}$$

where $V(\mathbf{x},\mathbf{y})$ is the visibility function defined as follows

$$V(\mathbf{x},\mathbf{y}) = \left\{ \begin{array}{ll} 1 & \text{if } \mathbf{x},\mathbf{y} \text{ are mutually visible} \\ 0 & \text{otherwise} \end{array} \right. \Bigg| \;\mathbf{x},\mathbf{y} \in \mathcal{M}\,. \tag{2.2}$$

Intuitively, we can interpret the geometric factor as projecting a very small area centered around $\mathbf{y}$ to the surface of the unit sphere placed at $\mathbf{x}$ and consecutively projecting the incurred solid angle to the tangent space $T_{\mathcal{M}}(\mathbf{x})$. The geometry factor is mainly used for transforming functions defined against the solid angle measure when integration is done with respect to the surface area measure.

### Ray space and throughput measure

We need to be familiar with the concept of the ray space [Vea98] $\mathcal{R}$ which is the Cartesian product $\mathcal{M} \times \mathcal{S}^2$. The ray $\mathbf{r}(\mathbf{x}, \overrightarrow{\omega}) \in \mathcal{R}$ starts at the surface point $\mathbf{x}$ and goes in the

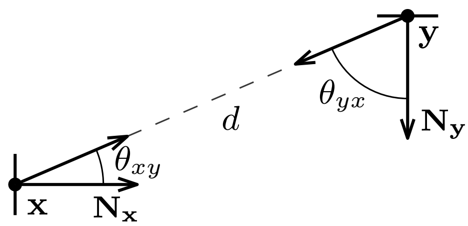


Figure 2.3: Geometric settings of two points

direction $\overrightarrow{\omega}$. The restriction to the set of points $\mathcal{M}$ emanates from the fact that there is no participating media present.

We next introduce the throughput measure [Vea98] $\tau$ on $\mathcal{R}$ in order to be able to integrate functions on the ray space. Its differential form is defined as

$$d\tau(\mathbf{r}) = d\tau(\mathbf{x}, \overrightarrow{\omega}) = dA(\mathbf{x}) d\sigma^{\perp}_{\mathbf{x}}(\overrightarrow{\omega})\,.$$

The throughput $\tau(D)$ of a set of rays $D \subseteq \mathcal{R}$ is then

$$\int_D d\tau(\mathbf{r}) = \int_D dA(\mathbf{x}) d\sigma^{\perp}_{\mathbf{x}}(\overrightarrow{\omega})\ .$$

The ray space can also be defined as the Cartesian product $\mathcal{M} \times \mathcal{M}$. A pair of points $(\mathbf{x}, \mathbf{y})$ defines the ray $\mathbf{r}(\mathbf{x}, \overrightarrow{\mathbf{xy}})$. The differential throughput measure of such ray is then

$$d\tau(\mathbf{r}) = dA(\mathbf{x}) \frac{d\sigma^{\perp}_{\mathbf{x}}(\overrightarrow{\mathbf{xy}})|\cos\theta_{\mathbf{yx}}|}{\|\mathbf{x}-\mathbf{y}\|^2} V(\mathbf{x}, \mathbf{y}) = dA(\mathbf{x}) dA(\mathbf{y}) G(\mathbf{x}, \mathbf{y})\ .$$

Note that defining $\mathcal{R}$ as the Cartesian product of all surface points leads to some redundancy. Two pairs of points $(\mathbf{x_1}, \mathbf{x_2})$ and $(\mathbf{x_1}, \mathbf{x_3})$ define the same ray as long as $\overrightarrow{\mathbf{x_1 x_2}} = \overrightarrow{\mathbf{x_1 x_3}}$. Nevertheless this formulation is used later when the *path integral formulation of light transport* is discussed.

### Other definitions

**Inner product** Given two real valued functions $f$, $g$ defined on a measurable space $(\mathbb{X}, \Sigma)$, we define the inner product of mentioned functions with respect to a measure $\mu$ defined on our measurable space as

$$\langle f, g\rangle = \int_{\mathbb{X}} f(x) g(x)\, d\mu(x)\ . \tag{2.3}$$

Inner product greatly simplifies the mathematical formulation of light transport.

**Dirac's delta function** Dirac's delta function is an important concept of the real analysis. It is real valued function denoted as $\delta(x)$ with the following properties:

1. $\delta(x) = \left\{ \begin{array}{ll} \infty & \text{if } x = 0 \\ 0 & \text{otherwise} \end{array} \right| \; x \in \mathbb{R}$

2. $\int_{\mathbb{R}} \delta(x)dx = 1$

Most of the time, we will be dealing with the Lebesgue integral of a real valued function defined on a measure space $\langle \mathbb{X}, \Sigma, \mu \rangle$, therefore we introduce the function $\delta_\mu^{\mathbf{x}'}$ for a fixed $\mathbf{x}' \in \mathbb{X}$ which denotes the Dirac's delta function of the given measure space. Analogously, $\delta_\mu^{\mathbf{x}'}$ has the following properties:

1. $\delta_\mu^{\mathbf{x}'}(\mathbf{x}) = \left\{ \begin{array}{ll} \infty & \text{if } \mathbf{x} = \mathbf{x}' \\ 0 & \text{otherwise} \end{array} \right| \mathbf{x}, \mathbf{x}' \in \mathbb{X}$

2. $\int_{\mathbb{X}} \delta_\mu^{\mathbf{x}'}(\mathbf{x}) d\mu(x) = 1 \mid \mathbf{x}, \mathbf{x}' \in \mathbb{X}$

Note that in the definition of delta function against the measure space we could simply use the *identity element* of that measure space, which would result in the definition very similar to $\delta(x \in \mathbb{R})$ (2.1.1). The approach taken will be justified later when we discuss specular BSDF or certain types of light sources.

### 2.1.2 Abstract model of light

One doesn't have to understand the exact physics of light to be able to create realistically looking computer images. Light is a synonym for electromagnetic radiation, which is carried by particles called photons. These particles have very complex behavior which differs by their wavelength. The human visual system is stimulated by photons that carry electromagnetic radiation at wavelength of range from 380 nm to 720 nm. We call this range the visible light. When simulating the visible light for the purposes of digital image synthesis we can afford many simplifications. That means we can drop certain details and use radiometric quantities as well as assume that the visible light travels on a straight line and use geometrical optics to transport these quantities in the space. This way we are still able to simulate important illumination effects and create digital images that are indistinguishable from photographs taken by today's cameras.

Now it is a good time to mention that throughout this thesis we will be dealing with physical measurements only. Human perception of these measurements is not discussed. That means we won't talk about photometry or colorimetry. The first chapter of the Sharma's book [Sha03] provides a comprehensive overview of the mentioned field.

### 2.1.3 Radiometry and radiometric quantities

Radiometry is a part of optics that deals with the measurement of electromagnetic radiation in the space. It is an abstraction from the actual photons. Because the number of photons required to stimulate sensors in the human visual system is very large, we no more treat photons as discreet particles, but instead we care about the energy $E$ they carry, which is given by $E = h\frac{c}{\lambda}$, where $h$ is the Planck constant and $c$ is the speed of light in vacuum. That means we regard the energy of photons as a continuous variable and we define the behavior of the visible light using mathematical tools for continuous variables. In the following text, radiometric quantities are defined and their intuitive meaning is given. However, in order to be able to develop global illumination algorithms, it is necessary to have solid understanding of radiometry in general.

### Phase space

Phase space $\psi$ [Vea98] is the Cartesian product $\mathbb{R}^3 \times \mathcal{S}^2 \times \mathbb{R}^+$, where individual parts stand for position, direction, and wavelength. It is a 6-dimensional space which can be used to describe the state of a photon particle at a fixed time. By counting the number of photons within a given region of the path space we acquire the photon number $N_p$ for that region. Using $N_p$ we can derive the function $f_d(\mathbf{p})$, which gives us the density of photon particles at the path space point $\mathbf{p}(\mathbf{x}, \overrightarrow{\omega}, \lambda)$. We also define the measure $\mu_\psi(\mathbf{p})$ on the phase space through its differential form

$$d\mu_\psi(\mathbf{p}) = d\mu_\psi(\mathbf{x}, \overrightarrow{\omega}, \lambda) = dA(\mathbf{x})d\sigma(\overrightarrow{\omega})d\lambda \tag{2.4}$$

### Radiant energy

Radiant energy $Q$ with units of $[J]$ is a function of time defined as

$$Q(t) = \int_{t_0}^{t} \int_{\mathbf{p}\in D} f_d(\mathbf{p}) d\mu_\psi(\mathbf{p}) dt \ , \tag{2.5}$$

where $D \subseteq \psi$.

Differentiation of $Q$ with respect to wavelength results in the spectral radiance energy $Q_\lambda$ with units of $[J \cdot m^{-1}]$.

### Radiant power (Radiant flux)

Radiant power or radiant flux $\Phi$ is defined as the derivative of radiant energy with respect to the time.

$$\Phi(t) = \frac{dQ(t)}{dt} \tag{2.6}$$

It has units of $[W]$. Intuitively, radiant flux represents the change of radiant energy at a fixed time within a subset of phase space. Like in the case of the spectral energy there exist spectral variant denoted $\Phi_\lambda$.

### Radiance

Radiance $L$ with units of $[W \cdot sr^{-1} \cdot m^{-2}]$ is the most important radiometric quantity in the context of global illumination. It is the amount of energy flowing through a point in a specific direction at a fixed time. More formally it can be defined using ray space and throughput measure as

$$L(\mathbf{r}) = \frac{d\Phi(\mathbf{r})}{d\tau(\mathbf{r})} \ , \tag{2.7}$$

where $\mathbf{r} \in \mathcal{R}$.

The following definition of $L$ is equivalent to 2.7 and is used more frequently within the global illumination literature.

$$L(\mathbf{x}, \overrightarrow{\omega}) = \frac{d\Phi(\mathbf{x}, \overrightarrow{\omega})}{\cos\theta d\sigma(\overrightarrow{\omega})dA(\mathbf{x})} = \frac{d\Phi(\mathbf{x}, \overrightarrow{\omega})}{d\sigma^\perp_{\mathbf{x}}(\overrightarrow{\omega})dA(\mathbf{x})} \tag{2.8}$$

In 2.8 $\mathbf{x} \in \mathbb{R}^3$, $\overrightarrow{\omega} \in \mathcal{S}^2$ and $\theta$ denotes the angle between the direction $\overrightarrow{\omega}$ and the surface normal $\mathbf{N_x}$. Note the $\cos\theta$ factor in 2.8 which tells that the radiance is defined with respect to the hypothetical differential surface centered around a point perpendicular to a direction of the radiance flow. Without this, the radiance would be dependent on a particular scene geometry.

Radiance is a function of time which is not stated explicitly to simplify notations. We can mostly disregard it. Recall that we only care about equilibrium radiance distribution within a scene. There is a slight problem with this simplification when dealing with light surface interaction. We need a way to distinguish between incoming (incident) and reflected (outgoing, leaving) radiance. This is relatively no issue when we are dealing only with reflective materials. Arvo [Arv95] introduced the *field radiance* at a fixed point as the radiance approaching from the set of directions $\mathcal{H}^2_-(\mathbf{x})$ and the *surface radiance* as the radiance leaving a fixed point in the set of directions $\mathcal{H}^2_+(\mathbf{x})$. If we want to simulate the effect of refraction, we need to distinct between incoming and reflected radiance as follows. Given a specific point in time $t$, we define the incident radiance $L_i$ as the state of the radiance field at $t$ and we define the outgoing radiance $L_o$ as the state of the radiance field at the limit point $t + \delta$.

Throughout the thesis, when we talk about incident radiance $L_i$ from a direction $\overrightarrow{\omega_i}$, we mean that light actually flows opposite the $\overrightarrow{\omega_i}$. This is a very common habit which for example simplifies definitions of reflectance functions.

#### Irradiance and radiant exitance

Irradiance $E$ with units of $[W \cdot m^{-1}]$ is defined as the total amount of energy arriving at a specific point $\mathbf{x}$ from all directions. We can define it using incident radiance as

$$E(\mathbf{x}) = \int_{\overrightarrow{\omega_i} \in \mathcal{S}^2} L_i(\mathbf{x}, \overrightarrow{\omega_i}) |\mathbf{N_x} \cdot \overrightarrow{\omega_i}| \, d\sigma(\overrightarrow{\omega_i}) = \int_{\overrightarrow{\omega_i} \in \mathcal{S}^2} L_i(\mathbf{x}, \overrightarrow{\omega_i}) \, d\sigma^{\perp}_{\mathbf{x}}(\overrightarrow{\omega_i}) \tag{2.9}$$

We can also define the irradiance as the derivative of $\Phi$ with respect to the surface area measure.

Naturally, we define the radiant exitance $B$, which stands for the total energy leaving a particular surface point $\mathbf{x}$ in all directions. It has the same units as irradiance and it is defined using outgoing radiance as

$$B(\mathbf{x}) = \int_{\overrightarrow{\omega_o} \in \mathcal{S}^2} L_o(\mathbf{x}, \overrightarrow{\omega_o}) |\mathbf{N_x} \cdot \overrightarrow{\omega_o}| \, d\sigma(\overrightarrow{\omega_o}) = \int_{\overrightarrow{\omega_o} \in \mathcal{S}^2} L_o(\mathbf{x}, \overrightarrow{\omega_o}) \, d\sigma^{\perp}_{\mathbf{x}}(\overrightarrow{\omega_o}) \tag{2.10}$$

### 2.1.4 Light surface interaction

Now that we have discussed the basic mathematical framework and the radiometry, we continue by defining the meaning of light surface interaction in the context of global illumination algorithms presented in this thesis. Recall that we represent the scene surface as a set of piecewise differentiable 2-manifolds (components) for practical reasons (memory consumption etc.). If we model the light as a large amount of discreet particles, we would have to break down the scene geometry to many components in order to achieve some detail. For this reason we model the light using radiometric quantities and describe its interaction with a surface statistically using functions.

#### 2.1.4.1 BRDF

*Bidirectional reflectance distribution function* (BRDF) is a four dimensional function which statistically describes how the light particles are reflected of the surface. Given a fixed point $\mathbf{x} \in \mathcal{M}$, the incident radiance in a direction $\overrightarrow{\omega_i} \in \mathcal{H}_+^2(\mathbf{x})$ and the outgoing radiance in a direction $\overrightarrow{\omega_o} \in \mathcal{H}_+^2(\mathbf{x})$, BRDF $f_r$ is defined as

$$f_r(\mathbf{x}, \overrightarrow{\omega_i}, \overrightarrow{\omega_o}) = \frac{dL_o(\mathbf{x}, \overrightarrow{\omega_o})}{d\sigma_{\mathbf{x}}^{\perp}(\overrightarrow{\omega_i}) L_i(\mathbf{x}, \overrightarrow{\omega_i})} = \frac{dL_o(\mathbf{x}, \overrightarrow{\omega_o})}{dE(\mathbf{x}, \overrightarrow{\omega_i})} \,. \tag{2.11}$$

Note that both directions in the previous equation are restricted to point to the upper hemisphere placed at the point $\mathbf{x}$, which implies that this function cannot be used to describe the transmission. For any physically valid BRDF following properties must hold.

1. $f_r(\mathbf{x}, \overrightarrow{\omega_i}, \overrightarrow{\omega_o}) = f_r(\mathbf{x}, \overrightarrow{\omega_o}, \overrightarrow{\omega_i})$ (Helmholtz's reciprocity)
2. $\int_{\overrightarrow{\omega_o} \in \mathcal{H}_+^2(\mathbf{x})} f_r(\mathbf{x}, \overrightarrow{\omega_i}, \overrightarrow{\omega_o})\, d\sigma_{\mathbf{x}}^{\perp}(\overrightarrow{\omega_o}) \leq 1$ (Energy conservation)
3. $f_r(\mathbf{x}, \overrightarrow{\omega_i}, \overrightarrow{\omega_o}) \geq 0$ (Non negativity)

The second property can be derived from the definitions of irradiance and radiant exitance.

**Diffuse reflection** BRDF of diffuse reflection, denoted $f_{r,d}$, is one of the simplest examples. We just want to scatter the $k_d \in (0, 1]$ portion of incident energy to all directions uniformly and obey properties of physically plausible BRDF, see figure 2.4. It is obvious that the resulting function has to be constant independent of directions proportional to $k_d$, so we only need to consider the conservation of energy. We express the amount of reflected energy using our unknown constant reflection function.

$$\int_{\overrightarrow{\omega_o} \in \mathcal{H}_+^2(\mathbf{x})} f_{r,d}\, d\sigma_{\mathbf{x}}^{\perp}(\overrightarrow{\omega_o}) = f_{r,d} \int_{\overrightarrow{\omega_o} \in \mathcal{H}_+^2(\mathbf{x})} d\sigma_{\mathbf{x}}^{\perp}(\overrightarrow{\omega_o}) = f_{r,d}\pi \tag{2.12}$$

We want the previous result to be equal to our constant $k_d$ which implies that BRDF for diffuse reflection is $f_{r,d} = \frac{k_d}{\pi}$.

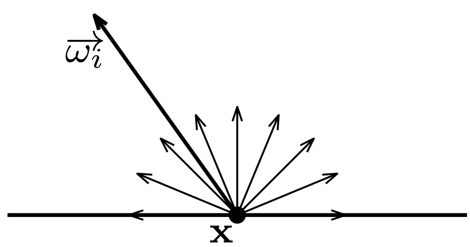


Figure 2.4: Incident radiance from the direction $\overrightarrow{\omega_i}$ is scattered uniformly to all directions.

**Specular reflection** BRDF for specular reflection is yet another simple example. In this case we want to scatter the incident radiance from a direction $\overrightarrow{\omega_i}$ only in the direction that corresponds to the reflection of $\overrightarrow{\omega_i}$ around the surface normal at a fixed point $\mathbf{x}$, see figure 2.5. As in the previous case we want to control the amount of reflected energy. We do this by the constant $k_r \in (0, 1]$. The specular reflection function is then defined as

$$f_{r,s}(\mathbf{x}, \overrightarrow{\omega_i}, \overrightarrow{\omega_o}) = k_r \delta_{\sigma_{\mathbf{x}}^{\perp}}^{\mathbf{R_x}(\overrightarrow{\omega_i})}(\overrightarrow{\omega_o}) \; . \tag{2.13}$$

Recall the definition of Dirac's delta function in the context of a measure space which can be used to show that following equation holds.

$$\int_{\overrightarrow{\omega_o} \in \mathcal{H}_+^2(\mathbf{x})} f_{r,s}(\mathbf{x}, \overrightarrow{\omega_i}, \overrightarrow{\omega_o}) d\sigma_{\mathbf{x}}^{\perp}(\overrightarrow{\omega_o}) = k_r$$

We see that $f_{r,s}$ defined as 2.13 truly reflects $k_r$ portion of the incident energy into the specular direction.

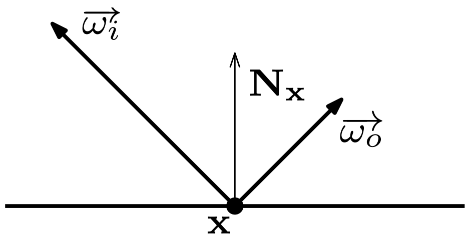


Figure 2.5: Portion of the incident radiance from the direction $\overrightarrow{\omega_i}$ is scattered in the perfect reflection direction only.

#### 2.1.4.2 BTDF

*Bidirectional transmission distribution function* (BTDF), denoted $f_t$, is the complement of the BRDF, because it describes the transmission of light energy. It can be non-zero only if directions $\overrightarrow{\omega_i}, \overrightarrow{\omega_o}$ point to different hemispheres centered at a point $\mathbf{x}$. More formally

$$f_t(\mathbf{x}, \overrightarrow{\omega_i}, \overrightarrow{\omega_o}) > 0 \Rightarrow (\mathbf{N_x} \cdot \overrightarrow{\omega_\mathbf{i}})\,(\mathbf{N_x} \cdot \overrightarrow{\omega_\mathbf{o}}) < 0.$$

Physically valid BTDF must obey properties analogous to BRDF but there is a slight difference. Helmholtz's reciprocity does not hold. Physical BTDF obey the following more general property

$$\frac{f_t(\mathbf{x}, \overrightarrow{\omega_i}, \overrightarrow{\omega_o})}{\eta_o^2} = \frac{f_t(\mathbf{x}, \overrightarrow{\omega_o}, \overrightarrow{\omega_i})}{\eta_i^2} \quad ,$$

where $\eta_i$ and $\eta_o$ are the refractive indices of materials containing $\overrightarrow{\omega_i}$ and $\overrightarrow{\omega_o}$ respectively.

Another important fact to mention is that when the light is refracted between materials with different indices of refraction, we have to scale the incident radiance by the factor $\frac{\eta_o^2}{\eta_i^2}$, where $\eta_i$ is the refraction index of the material to which the light is transmitted and $\eta_i$ is the refraction index of the material from which the light is incident.

**Specular transmission** Now we can derive the BTDF for the specular transmission. As with reflection, we want to transmit only $k_t \in (0, 1]$ amount of energy, see figure 2.6. Using Dirac's delta function and the fact discussed in the previous paragraph we can write BTDF for specular transmission as

$$f_{t,s}(\mathbf{x}, \overrightarrow{\omega_i}, \overrightarrow{\omega_o}) = k_t \frac{\eta_o^2}{\eta_i^2} \delta_{\sigma_{\mathbf{x}}^{\perp}}^{\mathbf{T}_{\mathbf{x}}(\overrightarrow{\omega_i})}(\omega_o) \ . \tag{2.14}$$

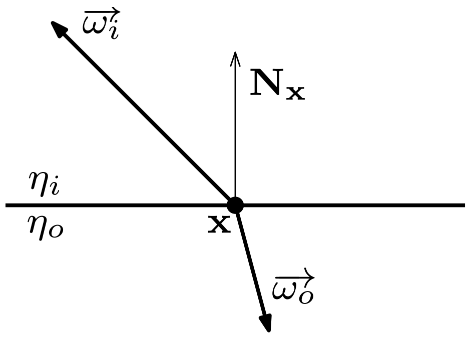


Figure 2.6: Portion of the incident radiance from the direction $\overrightarrow{\omega_i}$ is scattered in the perfect refraction direction only.

#### 2.1.4.3 BSDF

BSDF, denoted $f_s$, stands for *bidirectional scattering distribution function*. It is the union of BRDF and BTDF, which means that it describes both, the reflection and the transmission at a surface point. The reason we started with definitions of $f_r$ and $f_t$ is that we mostly model the materials as a combination of various BRDFs and BTDFs. Physically plausible BSDF has following properties. The only real difference from BRDF properties is in the reciprocity and in the domain of the integral used to formulate the property of energy conservation. They are provided here for reference.

1. $\frac{f_s(\mathbf{x}, \overrightarrow{\omega_i}, \overrightarrow{\omega_o})}{\eta_o^2} = \frac{f_s(\mathbf{x}, \overrightarrow{\omega_o}, \overrightarrow{\omega_i})}{\eta_i^2}$ (Reciprocity)
2. $\int_{\overrightarrow{\omega_o} \in \mathcal{S}^2} f_s(\mathbf{x}, \overrightarrow{\omega_i}, \overrightarrow{\omega_o})\, d\sigma_{\mathbf{x}}^{\perp}(\overrightarrow{\omega_o}) \leq 1$ (Energy conservation)
3. $f_s(\mathbf{x}, \overrightarrow{\omega_i}, \overrightarrow{\omega_o}) \geq 0$ (Non negativity)

**Adjoint BSDF**

Recall that Helmholtz's reciprocity does not hold for all physically valid BSDFs. For this reason we introduce the adjoint BSDF $f_s^*$ as

$$f_s^*(\mathbf{x}, \overrightarrow{\omega_i}, \overrightarrow{\omega_o}) = f_s(\mathbf{x}, \overrightarrow{\omega_o}, \overrightarrow{\omega_i}) \ , \tag{2.15}$$

which is used during the importance transport (discussed in 2.1.5) or particle tracing. The main advantage is that asjoint BSDF let us formulate a measurement using importance transport in the same way we do it using radiance transport, because directions $\overrightarrow{\omega_i}, \overrightarrow{\omega_o}$ have the same meaning in both cases.

Note that any reflectance function (BRDF) is self adjoint. The problem is when BSDF includes transmission. In that case, we have to formulate the adjoint explicitly. For example, using general reciprocity property, we can derive the adjoint specular transmission BSDF following way.

$$\begin{aligned} f_s^*(\mathbf{x}, \overrightarrow{\omega_i}, \overrightarrow{\omega_o}) &= f_{s,t}(\mathbf{x}, \overrightarrow{\omega_o}, \overrightarrow{\omega_i}) \\ &= \frac{\eta_i^2}{\eta_o^2} f_{s,t}(\mathbf{x}, \overrightarrow{\omega_i}, \overrightarrow{\omega_o}) \\ &= k_t \delta_{\sigma_{\mathbf{x}}^{\perp}}^{\mathbf{T_x}(\overrightarrow{\omega_i})}(\overrightarrow{\omega_o}) \end{aligned} \tag{2.16}$$

Note that in 2.15 there is no scaling by the factor $\frac{\eta_o^2}{\eta_i^2}$. This is an important detail.

### 2.1.5 Light transport and measurement

Throughout this section we precisely define a pixel measurement, the fundamental operation in digital image synthesis. Light sources and sensors (e.g. camera lens) are given in advance as a piece of the scene description. We discuss how radiance emitted by the light sources is transported through the environment and finally measured on camera sensor.

In the context of this thesis our sensor is a single pixel but it can be an arbitrary surface in the scene. We say that sensor emit importance $W_e$ which varies with position and direction. It is defined as

$$W_e(\mathbf{r}) = \frac{dS(\mathbf{r})}{d\Phi(\mathbf{r})} \qquad [S \cdot W^{-1}] \,, \tag{2.17}$$

where $\mathbf{r} \in \mathcal{R}$ and $S$ is the unit of sensor response. In the case of image synthesis $W_e$ is called *exitant importance function* and it has units of $[W^{-1}]$, which implies that corresponding response is unitless.

Having defined importance function, we can formulate pixel measurement $I$ as

$$I = \int_{\mathcal{R}} W_e(\mathbf{r}) L_i(\mathbf{r}) \, d\tau(\mathbf{r}) \,. \tag{2.18}$$

Note that 2.18 is simply the inner product of functions $W_e$ and $L_i$ on the ray space so we can simply write

$$I = \langle W_e, L_i \rangle \,.$$

As mentioned, $W_e$ is given to us in advance with the scene description but $L_i$ has to be computed through radiance propagation. Our goal is to express the incident radiance using emitted radiance function $L_e$, which will lead us to the solution of the pixel measurement.

The operator framework used in the following text was introduced by Erik Veach in his dissertation thesis [Vea98]. Our talk is limited only to the subset required to understand later chapters. We refer the interested reader to the original work, if she wants to gain more generic overview of measurement formulations.

### The propagation operator

We start by defining the propagation operator $\mathbf{G}$, which operates on functions defined on the ray space. This operator is based on ray casting function $\mathbf{x}_{\mathcal{M}}$ which is defined as

$$\mathbf{x}_{\mathcal{M}}(\mathbf{r}) = \mathbf{x}_{\mathcal{M}}(\mathbf{x}, \overrightarrow{\omega}) = \mathbf{x} + d_{\mathcal{M}}(\mathbf{x}, \overrightarrow{\omega})\, \overrightarrow{\omega} \;\; | \, \mathbf{x} \in \mathcal{M}, \overrightarrow{\omega} \in \mathcal{S}^2 \,, \tag{2.19}$$

where $d_{\mathcal{M}}(\mathbf{x}, \overrightarrow{\omega})$ is the *boundary distance function*, which gives the distance to the first point visible from $\mathbf{x}$ in the direction $\overrightarrow{\omega}$. Having precise definition of the ray casting, we continue with definition of $\mathbf{G}$.

$$(\mathbf{G}h)(\mathbf{x}, \overrightarrow{\omega}) = \begin{cases} h(\mathbf{x}_{\mathcal{M}}(\mathbf{x}, \overrightarrow{\omega}), -\overrightarrow{\omega}) & \text{if } d_{\mathcal{M}}(\mathbf{x}, \overrightarrow{\omega}) < \infty \\ 0 & \text{otherwise} \end{cases} \tag{2.20}$$

Now we can express the incident radiance at a point $\mathbf{x}$ from direction $\overrightarrow{\omega_i}$ in terms of the outgoing radiance from another point in the scene by writing

$$L_i(\mathbf{x}, \overrightarrow{\omega_i}) = (\mathbf{G}L_o)(\mathbf{x}, \overrightarrow{\omega_i}).$$

Or we can simply express $L_i$ using $L_o$ as $L_i = \mathbf{G}L_o$.

### The scattering operator

We define another operator on the ray space, the scattering operator $\mathbf{K}$, as

$$(\mathbf{K}h)(\mathbf{x}, \overrightarrow{\omega_o}) = \int_{\overrightarrow{\omega_i} \in \mathcal{S}^2} f_s(\mathbf{x}, \overrightarrow{\omega_i}, \overrightarrow{\omega_o}) h(\mathbf{x}, \overrightarrow{\omega_i})\, d\sigma^{\perp}_{\mathbf{x}}(\overrightarrow{\omega_i}) \,. \tag{2.21}$$

Note that by plugging the incident radiance function expressed using the propagation operator as $\mathbf{G}L_o$ into $\mathbf{K}$ we get the *local scattering equation*, which gives us the outgoing radiance from a fixed point in a certain direction due to illumination from the entire sphere centered at the point.

### The transport operator and the solution operator

To simplify further notations and derivations, we combine previously defined operators into a single one, the transport operator

$$\mathbf{T} = \mathbf{K}\mathbf{G} \,, \tag{2.22}$$

which allows us to formulate the exitant radiance from a point in a fixed direction as

$$L_o(\mathbf{x}, \overrightarrow{\omega_o}) = L_e(\mathbf{x}, \overrightarrow{\omega_o}) + (\mathbf{T}L_o)(\mathbf{x}, \overrightarrow{\omega_o}) \,. \tag{2.23}$$

Note that 2.23 is actually *the rendering equation* [Kaj86] formulated using operator framework.

The transport operator allows us to derive a solution to the rendering equation in the form of a single operator, the solution operator, denoted $\mathbf{S}$. The solution operator can be relatively easily derived from the operator formulation of the rendering equation 2.23 as

$$\begin{aligned} L_o &= L_e + \mathbf{T}L_o \\ (\mathbf{E} - \mathbf{T})L_o &= L_e \\ L_o &= (\mathbf{E} - \mathbf{T})^{-1} L_e \\ L_o &= \mathbf{S}L_e \end{aligned} \tag{2.24}$$

The solution operator exists if and only if we use physically valid BSDFs, in which case it can be written as

$$\mathbf{S} = L_e + \mathbf{T}L_e + \mathbf{T}^2 L_e + ... + \mathbf{T}^\infty L_e \ . \tag{2.25}$$

This formulation says that exitant radiance is equal to the sum of emitted radiance, emitted radiance scattered once, emitted radiance scattered twice etc.

#### Operator measurement formulation

We have formulated the necessary mathematical framework, which allows us to express the measurement 2.18 by the inner product of two functions defined on the ray space

$$I = \langle W_e, \mathbf{GS}L_e \rangle \ . \tag{2.26}$$

Note that using this form of pixel measurement $I$ and Monte Carlo techniques, discussed in the 2.1.6, we can derive the standard *path tracing* algorithm. The definition of the solution operator 2.25 directly serves as a guideline to the path tracing.

#### Importance transport

Naturally, we don't have to restrict our selves to the radiance transport only. We can imagine that our sensors emit importance in to the environment in the same way as the light sources emit radiance. Importance is scattered upon surface interaction and occasionally ends up on a light source, which now hypothetically figures as a sensor. We then have the incident importance function $W_i$ and exitant important function $W_o$. The meaning of these functions is analogous to the radiance counterparts. Now we can formulate a measurement as

$$I = \langle \mathbf{GS}W_e, L_e \rangle \ . \tag{2.27}$$

There is a slight issue with this formulation as it is only valid when the solution operator is self adjoint. This condition is met only if we use symmetric BSDFs (e.g. Helmholtz's reciprocity holds). In the case of asymmetric BSDFs, recall the specular transmission, we have to define the adjoint solution operator $\mathbf{S}^*$ explicitly using adjoint BSDFs.

### 2.1.6 Monte Carlo integration

This section provides a brief overview of the Monte Carlo methods used in the context of global illumination algorithms [Shi01b]. It discusses the basic principles of the Monte Carlo numerical integration together with techniques that have shown to be of practical use, when rendering photorealistic images. That said, this section doesn't contain thorough derivations and proofs of all stated theorems. References to relevant resources are provided wherever possible. It is assumed that the reader is already familiar with the theory of *continuous random variables* [Ros09].

#### 2.1.6.1 Introduction

The Monte Carlo is a type of numerical integration useful for estimating integrals on multidimensional domain. For example. recall that exitant radiance from a given point in a certain direction is, among others, given by an integral over the two dimensional space of directions $\mathcal{S}^2$. In 2.2 we even formulate a measurement using an infinite dimensional space of light carrying paths.

Let us have a function $g$ defined on a measure space $\langle \mathbb{X}, \Sigma, \mu \rangle$.

$$I = \int_{x \in \mathbb{X}} g(x) d\mu(x) \tag{2.28}$$

is the integral of $g$ over the whole measure space $\mathbb{X}$ with respect to the measure $\mu$. The Monte Carlo estimator for 2.28 looks like

$$\langle I \rangle = \frac{1}{N} \sum_{i=1}^{N} \frac{g(x_i)}{p(x_i)} . \tag{2.29}$$

In 2.29 $N \in \mathbb{N}$, the number of samples taken, and $x_i$ is a continuous random variable. All random variables $x_i$ are *identically independently distributed* (IID) according to *probability density function* $p$, which has the property

$$\int_{x \in \mathbb{X}} p(x) d\mu(x) = 1 .$$

The probability that a random variable $x \sim p$ takes a value from a subset $D \subseteq \mathbb{X}$ is

$$P(x \in D) = \int_{x' \in D} p(x') d\mu(x') . \tag{2.30}$$

Not that 2.29 is also an random variable and its expected value is equal to the value of the integral being estimated as its shown in 2.31.

$$E[\langle I \rangle] = E\left[\frac{1}{N} \sum_{i=1}^{N} \frac{g(x_i)}{p(x_i)}\right] = \frac{1}{N} \sum_{i=1}^{N} E\left[\frac{g(x_i)}{p(x_i)}\right] = \frac{1}{N} \sum_{i=1}^{N} \int_{x \in \mathbb{X}} \frac{g(x)}{p(x)} p(x) d\mu(x) = I \tag{2.31}$$

This also shows that the expected value of $\langle I \rangle$ is the sum of expected values of IID random variables $\frac{g(x_i)}{p(x_i)}$. Recall the definition of *variance* $V[x] = E\big([x - E(x)]^2\big)$. Using the *law of large numbers* it is relatively easy to see that

$$P\left(\lim_{N \to \infty} V\left[\frac{1}{N} \sum_{i=1}^{N} \frac{g(x_i)}{p(x_i)}\right] = 0\right) = 1 .$$

Previous statement says that by performing an infinite number of trials we get the value of the integral estimated as long as $p$ is non-zero whenever $g$ is. Of course, we would like to have close estimate after a smallest amount of trials as possible. Setting $N = \infty$ corresponds to performing the whole integration. When $x_i$ in the 2.29 are IID then the following derivation shows that to halve the variance we have to double the number of samples.

$$V\left[\frac{1}{N} \sum_{i=1}^{N} \frac{g(x_i)}{p(x_i)}\right] = \frac{\sum_{i=1}^{N} V\left[\frac{g(x_i)}{p(x_i)}\right]}{N^2} = \frac{V\left[\frac{g(x_i)}{p(x_i)}\right]}{N}$$

This result also implies that standard deviation $\sigma$ decreases with $\sqrt{N}$, which means to halve it we need to quadruple the number of samples.

#### 2.1.6.2 Importance sampling

As we discussed, the estimator 2.29 always converge to the value of integral being estimated if $g(x) > 0 \Rightarrow p(x) > 0$. Of course as long as $N < \infty$ there will be a non-zero variance of the estimator in practical scenarios (not necessarily always), which introduces itself as a noise. Imagine we pick our samples from the distribution

$$p_g(x) = \frac{g(x)}{\int_{x' \in \mathbb{X}} g(x') d\mu(x')} , \tag{2.32}$$

where $x \in \mathbb{X}$. In such case, we need only one sample to have the variance of zero, because the fraction $\frac{g(x)}{p_g(x)}$ is equal to the integral for any $x \in \mathbb{X}$. Of course, $p_g$ implies knowledge of $I$, so why would we use it to estimate $I$ anyway. The point is, that we can lower the variance of an estimator by making sure that $p(x)$ matches $g(x)$ as much as possible. This approach to variance reduction is called *importance sampling*. The problem of sampling a domain according to an arbitrary probability density function is discussed next.

**Canonical uniform random variable** Uniform continuous random variable can take on a value from an real interval $[a, b)$ with equal probability given by the constant probability density $\frac{1}{b-a}$. When the interval is $[0, 1)$ we talk about canonical uniform real random variable, which is denoted $\xi$, see figure 2.7. Note that the probability density for $\xi$ is equal to 1. In practice we are given a random number generator that provides us with $\xi$. We then have to transform the canonical distribution to the desired probability density.

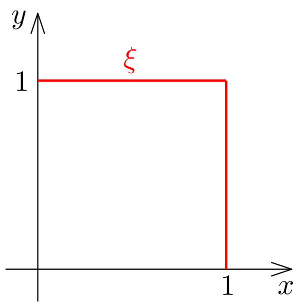


Figure 2.7: The canonical uniform random variable.

**Inversion method**

The inversion method is a way of sampling an interval of real numbers according to some probability density. Say we have two continuous random variables $x \sim p_x$ and $y \sim p_y$ but we are only able to draw samples from $p_x$. Then we can define the random variable $y$ as $y = T(x)$. Now the problem is to find the transformation function $T$ so that the distribution of $y$ is exactly $p_y$ [PH10]. Function $T$ has to be a one-to-one mapping and its derivative has to be strictly positive. This fact leads to the relation of *cumulative density functions* (CDF)

$$P_y(y) = P_y(T(x)) = P_x(x) \ ,$$

which can be used to express $T$ as

$$T(x) = P_y^{-1}\big(P_x(x)\big) \ .$$

In a common case, $x$ is the canonical uniform random variable so

$$T(x) = P_y^{-1}(x) \ .$$

Now it should be obvious, why this methods is called the inversion method. We find the CDF of $p_y$, we invert it and we use it to directly transform samples from $\xi$. If $P_y^{-1}$ is not invertible, we can the rejection method, which is not discussed here. Figure 2.8 visualizes the process of mapping the canonical continuous random variable to the random variable $y \sim p_y$.

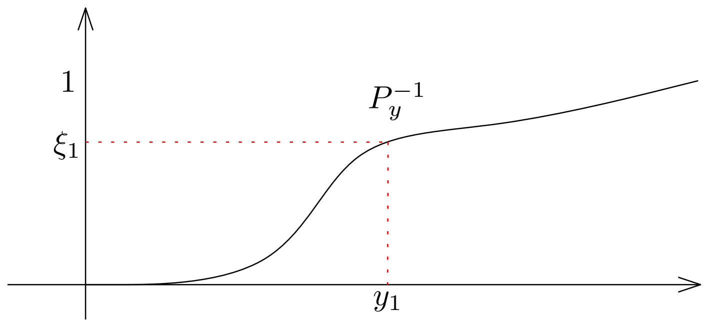


Figure 2.8: Visualization of the inversion method. The random variable $\xi_1$ is mapped to the random variable $y \sim p_y$ through $P_y^{-1}$.

### Multiple dimensions

Suppose we want to sample a certain subset of $\mathbb{R}^2$ given a density function $p(x, y)$. We derive the *marginal probability density*

$$p(x) = \int_{\mathbb{R}} p(x, y)\, dy \tag{2.33}$$

and the *conditional density*

$$p(y|x) = \frac{p(x, y)}{\int_{\mathbb{R}} p(x, y) dy} = \frac{p(x, y)}{p(x)} \ . \tag{2.34}$$

It is relatively easy to show that $p(x)$ and $p(y|x)$ are probability densities. We now generate samples according to $p(x, y)$ by sampling 2.33 and 2.34 separately using techniques for single dimension (e.g. the inversion method) [PH10].

As an example, consider we want to sample the $\mathcal{H}^2_+(\mathbf{x})$ set of direction and we have a probability density with respect to the solid angle measure $\sigma$. Solid angle measure is useful but very abstract, so we use the $(\theta, \phi)$ parametrization of the directions. The solid angle produced by a fixed point $(\theta, \phi)$ is

$$d\sigma = \sin\theta d\theta d\phi \ ,$$

which leads us to the relation of the density functions

$$p(\theta, \phi) = p(\omega) \sin\theta \ .$$

So given a density $p(\omega)$ we derive the density $p(\theta, \phi)$ and we use the technique using marginal and conditional density described above.

#### 2.1.6.3 Multiple importance sampling

Multiple importance sampling [Vea98] (MIS) is a very useful variance reduction technique that is used extensively in global illumination. Recall the scattering equation 2.23 which includes the integral of the product of multiple functions, namely $L_i$ and $f_s$. We can sample it using the density function that matches BSDF but if the incident radiance would be large from unimportant directions according to the BSDF probability density, variance would also be large. Same applies if somehow we know, how to sample directions using a density function that roughly matches the incident radiance function (e.g. by using photon map). Multiple importance sampling is about combining samples from different sampling techniques (density functions), so that variance is lowered.

Say, we want to estimate an integral of function $g$ defined on an measure space $\langle \mathbb{X}, \Sigma, \mu \rangle$

$$I = \int_{\mathbb{X}} g(x) d\mu(x)$$

and we have $n$ different sampling techniques, where $p_i$ denotes the probability density of $i$-th technique. Each technique is good for sampling a certain subset of $\mathbb{X}$ but not good for sampling the whole domain. If we take $n_i \geq 1$ samples from each technique, then we can combine them using the *multi-sample estimator* [Vea98]

$$F[I] = \sum_{i=1}^{n} \frac{1}{n_i} \sum_{j=1}^{n_i} w_i(x_{ij}) \frac{f(x_{ij})}{p_i(x_{ij})} \ , \tag{2.35}$$

where $w_i$ is the weighting function for samples from $i$-th sampling technique. If we want $F$ to produce an unbiased estimate of $I$, the weighting functions must satisfy following conditions.

1. $\sum_{i=1}^{n} w_i(x) = 1$ iff $f(x) \neq 0$
2. $w_i(x) = 0$ whenever $p_i(x) = 0$

There are many weighting functions that satisfies these conditions but only a few helps to lower the variance. In general case, Veach showed [Vea98] that the best strategy one can use is the *balance heuristic*, which implies usage of weighting functions defined as

$$w_i(x) = \frac{n_i p_i(x)}{\sum_{k=1}^{n} n_k p_k(x)} \ . \tag{2.36}$$

Sometimes it might be better to use more general version of balance heuristic, called the *power heuristic* with weighting functions 2.37.

$$w_i(x) = \frac{\left(n_i p_i(x)\right)^{\beta}}{\sum_{k=1}^{n} \left(n_k p_k(x)\right)^{\beta}} , \tag{2.37}$$

where $\beta \in \mathbb{R}^+$ (usually $\beta = 2$). Note that the balance heuristic is the special case of the power heuristic. For a discussion when the power heuristics is practical, we refer the interested reader to the original work [Vea98].

#### 2.1.6.4 Russian roulette

The last Monte Carlo technique we discuss is Russian roulette which allows us to estimate recursive integrals or integrals over infinite dimensional spaces without introducing bias. The idea is rather very simple. We assign a non-zero discreet probability $q$ to sampling element with zero measure. The original probability density then has to be scale by the factor $1-q$.

## 2.2 Path sampling methods

In this section we start by reviewing the *path integral formulation of light transport* (*path integral framework*), which allows us to write a pixel measurement $I$ in the form of a non-recursive integral equation. Recall that the rendering equation contains a recursive integral. This formulation was introduced by Veach in his dissertation thesis [Vea98] and it serves as a good basis for formulating unbiased global illumination algorithms presented in this section.

### 2.2.1 Path space formulation of light transport

As mentioned, *path space formulation of light transport* is presented in a very formal and rigorous way. The following text should serve as a comprehensive overview of important definitions required to formulate global illumination algorithms based on path sampling. We refer the interested reader to the original work [Vea98], if she wants to know more details.

We start by a formal definition of a geometric primitive called *path* and proceed further towards the formulation of pixel measurement $I$ on the *path space*.

**Path**

A path (figure 2.9) of length $k \geq 0$ is a $(k+1)$-tuple of surface points

$$\overline{x} = (\mathbf{x}_0, \ldots, \mathbf{x}_k) ,$$

where $\mathbf{x}_i \in \mathcal{M}$. It can also be defined as a member of the set

$$\Omega_k = \mathcal{M}^{k+1} ,$$

which contains all possible paths of length $k$. Note that by this definition we allowed the path points to be located only at the scene surfaces. Placing points anywhere in the space

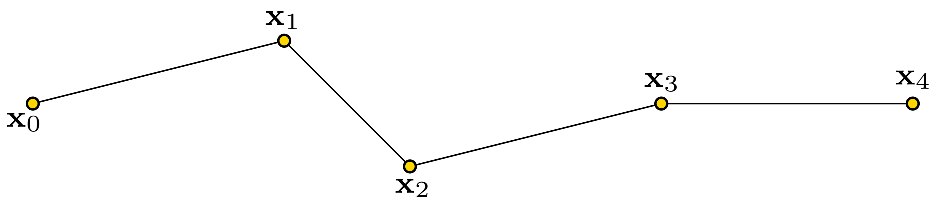


Figure 2.9: A path of length $k = 4$.

would be useless for our purposes. Recall that we are disregarding effects of participating media. Also note that every two adjacent points $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$ define the ray $\mathbf{r}(\mathbf{x}_i, \mathbf{x}_{i+1})$ that goes from $\mathbf{x}_i$ towards $\mathbf{x}_{i+1}$.

*Light path* is a path which has the first vertex located on the light source. It is assumed that every such path $\overline{x}$ implicitly contains imaginary vertex with the index of $-1$, $\mathbf{x}_{-1}$, and that

$$L_e(\mathbf{x}_{-1} \to \mathbf{x}_0) = L_e(\mathbf{x}_0) \tag{2.38}$$

$$f_s(\mathbf{x}_{-1} \to \mathbf{x}_0 \to \mathbf{x}_1) = \frac{L_e(\mathbf{x}_0 \to \mathbf{x}_1)}{L_e(\mathbf{x}_0)} \ , \tag{2.39}$$

where $L_e(\mathbf{x}_0)$ represents the total emitted radiance per squared meter at the point $\mathbf{x}_0$. The imaginary vertex can be thought as the source of all energy. This seemingly artificial definition will be useful later as it makes the formulations of path sampling algorithms short and compact.

*Eye path* is then naturally a path with the first vertex located on the camera sensor. Analogically, every eye path $\overline{x}$ contains imaginary vertex $\mathbf{x}_{-1}$ and

$$W_e(\mathbf{x}_{-1} \to \mathbf{x}_0) = W_e(\mathbf{x}_0) \tag{2.40}$$

$$f_s(\mathbf{x}_1 \to \mathbf{x}_0 \to \mathbf{x}_{-1}) = \frac{W_e(\mathbf{x}_0 \to \mathbf{x}_1)}{W_e(\mathbf{x}_0)} \ , \tag{2.41}$$

where $W_e(\mathbf{x}_0)$ is the total emitted importance per squared meter at the point $\mathbf{x}_0$.

### Path space

We naturally continue by defining the set of all paths of all possible lengths $k \geq 0$ to which we will refer to as the path space

$$\Omega = \sum_{k=1}^{\infty} \Omega_k$$

A single path is denoted $\overline{x}$ and has the length $k(\overline{x})$. Note that the path space is an infinite dimensional space.

### Area product measure

In order to integrate functions defined on the path space, we define the area product measure $\mu_A$ so that

$$\mu_A(\Omega) = \sum_{k=0}^{\infty} \mu_A(\Omega_k) = \sum_{k=0}^{\infty} A(\mathcal{M})^{k+1}$$

This way a path $\overline{x}$ of the length $k$ has the differential *area product measure*

$$d\mu_A(\overline{x}) = d\mu_A(\mathbf{x}_0, \ldots, \mathbf{x}_k) = \prod_{i=0}^{k} dA(\mathbf{x}_i)$$

### Measurement contribution function and throughput

Given a fixed path $\overline{x}$ of the length $k$, the measurement contribution function $f(\overline{x})$ gives us the energy flowing through the path per unit of the area product measure. We first define the path throughput $T(\overline{x})$ as

$$T(\overline{x}) = G(\mathbf{x}_0, \mathbf{x}_1) \prod_{i=1}^{k-1} f_s(\mathbf{x}_{i-1} \to \mathbf{x}_i \to \mathbf{x}_{i+1}) G(\mathbf{x}_i, \mathbf{x}_{i+1}) \ . \tag{2.42}$$

Using the definition 2.42 we can define the measurement contribution function as

$$f(\overline{x}) = L_e(\mathbf{x}_0 \to \mathbf{x}_1) T(\overline{x}) W_e(\mathbf{x}_k \to \mathbf{x}_{k-1}) \tag{2.43}$$

Note that in 2.43 $\to$ denotes the direction of the light flow when used in the context of $L_e$ and denotes the direction of the importance flow in the case of $W_e$. This notation is not in correspondence with Veach's original formulation. Also note that $f(\overline{x})$ is non-zero iff $\mathbf{x}_0$ lies on the light source and $\mathbf{x}_k$ lies on the camera lens. Let us denote the set off all such non-zero contribution paths $\Omega_f$.

Sometimes it will be useful to work with the eye path throughput of an eye path $\overline{x}$

$$T_E(\overline{x}) = W_e(\mathbf{x}_0) \prod_{i=1}^{k} f_s(\mathbf{x}_i \to \mathbf{x}_{i-1} \to \mathbf{x}_{i-2}) G(\mathbf{x}_{i-1}, \mathbf{x}_i) \ . \tag{2.44}$$

As in the case of the eye path, this definition is used to make the formulations of path sampling algorithms short and compact.

Analogically, we formulate the light path throughput

$$T_L(\overline{x}) = L_e(\mathbf{x}_0) \prod_{i=1}^{k} f_s(\mathbf{x}_{i-2} \to \mathbf{x}_{i-1} \to \mathbf{x}_i) G(\mathbf{x}_{i-1}, \mathbf{x}_i) \ . \tag{2.45}$$

### Pixel measurement

Using definitions presented so far, we can finally define the pixel measurement $I$ using the path space

$$I = \sum_{k=1}^{\infty} \int_{\Omega_k} f(\overline{x}) \, d\mu_A(\overline{x}) = \tag{2.46}$$

$$\int_{\Omega_f} f(\overline{x}) \, d\mu_A(\overline{x}) \tag{2.47}$$

Formulation of the pixel measurement given above is elegant and intuitive. Nevertheless it is not by far the most important advantage. As we can see in 2.43 there is no need for special treatment of non-symmetric *BSDF*, which makes formulations and implementations of bi-directional algorithms less error prone.

### Measurement estimation

We define the Monte Carlo estimator of 2.46

$$\langle I \rangle = \sum_{k=1}^{\infty} \frac{1}{N_k} \sum_{i=1}^{N_k} \frac{f(\overline{x}_i)}{p(\overline{x}_i)} \; . \tag{2.48}$$

In 2.48 $N_k$ denotes the number of samples taken to estimate the contribution of paths of length $k$ and $p(\overline{x})$ stands for probability density of path $\overline{x}$ with respect to the area product measure. We can easily show that 2.48 is an unbiased estimator of 2.46 and 2.47.

$$E[\langle I \rangle] = \sum_{k=1}^{\infty} \frac{1}{N_k} \sum_{i=1}^{N_k} E\left[\frac{f(\overline{x}_i)}{p(\overline{x}_i)}\right] = \sum_{k=1}^{\infty} \frac{1}{N_k} \sum_{i=1}^{N_k} \int_{\Omega_k} \frac{f(\overline{x})}{p(\overline{x})} p(\overline{x}) d\mu_A(\overline{x}) = \sum_{k=1}^{\infty} \int_{\Omega_k} f(\overline{x}) d\mu_A(\overline{x}) = I$$

Constructing $p(\overline{x})$ for a set of paths of fixed length $\Omega_k$ is straightforward. Recall that $\overline{x} \in \Omega_k$ is simply a $(k+1)$-tuple of surface points $(\mathbf{x}_0, \ldots, \mathbf{x}_k)$. Assuming that each point $\mathbf{x}_i$, $i \in [0, k]$, is selected using a *stationary distribution* $p_{A_i}(\mathbf{x}_i)$ with respect to the surface area measure, we can write $p(\overline{x})$ as

$$p(\overline{x}) = \prod_{i=0}^{k} p_{A_i}(\mathbf{x}_i) \; .$$

This construction of $p(\overline{x})$ gives us a way to perform importance sampling on global scale. We are not limited to doing it locally as with algorithms based on radiance/importance propagation. This is another great advantage of path integral formulation.

There are certain limitations on what paths can be sampled. If a path expressed using the *extended notation for light paths* does not contain the **DD** substring, then unbiased algorithms are unable to account for its contribution. For example, caustics from a point light seen through a mirror by a pinhole camera.

### 2.2.2 Path tracing

The first unbiased algorithm we discuss is the path tracing. It has been originally formulated by Kajiya as a solution to the *rendering equation* [Kaj86]

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_{\mathcal{S}^2} (\mathbf{G}L_o)(x, \omega_i) f_s(x, \omega_i, \omega_o) d\mu_x^{\perp}(\omega_i) \; , \tag{2.49}$$

where $\mathbf{G}$ is the propagation operator. He proposed an obvious recursive algorithm that solves 2.49. Note that the measurement formulation $\langle W_e, \mathbf{GS}L_e \rangle$ can be thought as an extension of the rendering equation, which allows as to measure the incident radiance on more than a one point from more than a one direction. We formulate path tracing using the path integral framework. The algorithm 1 outlines the basic structure of it. Various mutations only differ in the way the estimate of the radiance flowing from a surface point to a specific pixel sample point is computed.

**Algorithm 1** Path tracing algorithm skeleton

```
function PathTracing(pixelSamples)
    for i ← 1 to pixelCount do
        image[i] = 0
        for j ← 1 to pixelSamples do
            pixelSample = GenerateRandomPixelSample(i)
            ray = MapPixelSampleToRay(pixelSample)
            radiance = EstimateRadiance(ray)
            image[i] += radiance / pixelSamples
        end for
    end for
    return image
end function
```

Naive path tracing (unidirectional) is based on estimating the pixel measurement $I$ by sampling the path space using eye paths only. More formally the estimate can be written as

$$\langle I \rangle = \sum_{t=2}^{\infty} \frac{f(\overline{x}_t)}{p_t(\overline{x}_t)} \, , \tag{2.50}$$

where $\overline{x}_t$ denotes an eye path that consist of $t$ vertices and as the length $t-1$, $f$ is the measurement contribution function and $p_t$ is the probability density function of eye path with $t$ vertices.

Definition 2.50 is all but useful, because it contains the sum over infinite number of path samples. For efficiency reasons, we recursively generate only a single eye path $\overline{z}$ by a random walk trough the scene. Every non-empty subpath of $\overline{z}$ starting at $\mathbf{z}_0$ is also an eye path, so the original path represents correlated samples of eye paths with $t \leq k(\overline{z})$ vertices. The generation of $\overline{z}$ is possibly terminated using the Russian roulette technique at every new vertex after a certain number of vertices has already been generated. We can think of the Russian roulette in this context as sampling a *zero contribution vertex* $\epsilon$. After sampling $\epsilon$ there is no reason to continue, because paths of lengths $t > k(\overline{z})$ would have zero measurement contribution. Using our eye path $\overline{z}$, we can write the unbiased estimate of the pixel measurement as

$$\begin{aligned} \langle I \rangle &= \sum_{t=2}^{k(\overline{z})+1} \frac{f(\overline{z}_t)}{p_t(\overline{z}_t)} \\ &= \sum_{t=2}^{k(\overline{z})+1} \frac{T_E(\overline{z}_t)}{p_t(\overline{z}_t)} L_e(\mathbf{z}_{t-1} \to \mathbf{z}_{t-2}) \\ &= \sum_{t=2}^{k(\overline{z})+1} \alpha_t L_e(\mathbf{z}_{t-1} \to \mathbf{z}_{t-2}) \, , \end{aligned} \tag{2.51}$$

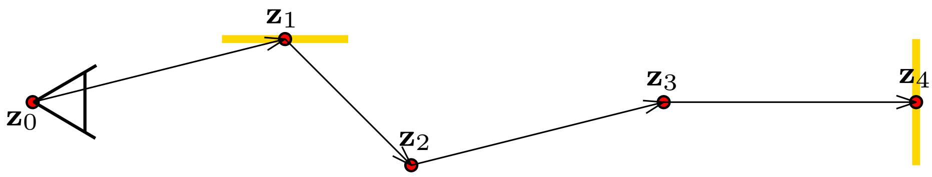


Figure 2.10: A single measurement during unidirectional path tracing.

where $\overline{z}_t$ denotes the subpath of $\overline{z}$ with vertices $(\mathbf{z}_0, \ldots, \mathbf{z}_{t-1})$ and $T_E(\overline{z})$ denotes its eye path throughput. As we can see from 2.51, to evaluate the measurement contribution of path with $t$ vertices, we just need to compute the emitted radiance from the vertex $\mathbf{z}_{t-1}$ towards the vertex $\mathbf{z}_{t-2}$. The coefficient $\alpha_t$ can be computed recursively during the random walk as

$$\alpha_2 = 1 \tag{2.52}$$

$$\alpha_t = \frac{f_s(\mathbf{z}_{t-1} \to \mathbf{z}_{t-2} \to \mathbf{z}_{t-3}) \cos\theta_{\mathbf{z}_{t-2}\mathbf{z}_{t-1}}}{p_\sigma(\mathbf{z}_{t-2} \to \mathbf{z}_{t-1})} \times \alpha_{t-1} \,, \tag{2.53}$$

where $p_\sigma$ denotes a probability density with respect to the solid angle measure.

Figure 2.10 depicts a single pixel measurement within the path tracing algorithm. Note that there is no contribution of paths of lengths 2 and 3. Non-zero probability is given to a large set of paths with zero contribution, which introduces high variance.

Unidirectional path tracing is not very efficient algorithm solving global illumination, because the probability density $p_t$ does not match the measurement contribution function $f(\overline{z}_t)$ very well in most cases. A simple variance reduction technique that doesn't make the path tracing much harder to implement is the *next event estimation*. For every eye subpath $\overline{z}_t$, where $t \geq 1$, we generate a light path $\overline{y}$ consisting of a single vertex, using probability density defined against the surface area measure. The measurement estimate is then

$$\begin{aligned}\langle I\rangle &= \sum_{t=1}^{k(\overline{z})+1} \frac{f(\overline{z}_t\overline{y})}{p_t(\overline{z}_t)p_A(\mathbf{y}_0)} \\ &= \sum_{t=1}^{k(\overline{z})+1} \frac{T_e(\overline{z}_t)}{p_t(\overline{z}_t)} \frac{f_s(\mathbf{y}_0 \to \mathbf{z}_{t-1} \to \mathbf{z}_{t-2})G(\mathbf{y}_0, \mathbf{z}_{t-1})L_e(\mathbf{y}_0 \to \mathbf{z}_{t-1})}{p_A(\mathbf{y}_0)} \\ &= \sum_{t=1}^{k(\overline{z})+1} \alpha_t^E\,\alpha^L \,,\end{aligned} \tag{2.54}$$

where coefficient $\alpha^L$ is evaluated upon connection of $\overline{z}_t$ to the generated light vertex and $\alpha_t^E$ is constructed recursively as in the case of unidirectional sampling.

In 2.54, the probability of successfully connecting the eye subpath $\overline{z}_1$ to the generated light vertex is very low. In the context of image synthesis, $I$ represents the measurement for a single pixel, so number of directions with zero contribution to measurement is relatively low. This leave us with two options. We can either give up on connecting the first vertex to the light source and account for visible light sources as in unidirectional path tracing or we can perform measurement for all pixels at once. In the second case, we have to weight measurement contribution for technique $t = 1$ by the factor $\frac{1}{N_{pixels}}$.

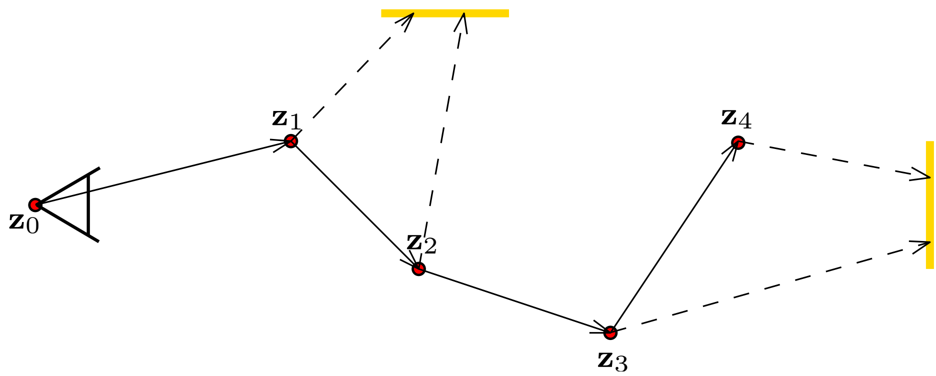


Figure 2.11: A single measurement during the path tracing with next event estimation without explicit connection to the camera vertex.

Figure 2.11 depicts a single pixel measurement in the context of the path tracing with the next event estimation. Dashed arrows represent the explicit connection of a path to the light source. Note that connection is not performed at the first eye path vertex.

There is another simple improvement of the path tracing with the next event estimation that lowers the variance when area light sources are present in the scene. It lies in using more than one sampling technique when generating a light point for an eye subpath $\overline{z}_t$. In 2.54, we are generating a light point using a probability density defined directly against the surface area measure. We can also try to generate a light point by sampling the BSDF at vertex $\mathbf{z}_{t-1}$ and combine results from both techniques using multiple importance sampling.

### 2.2.3 Bidirectional path tracing

Bidirectional path tracing [Vea98][VG94][LW93] can be thought as a generalization of the path tracing with the next event estimation. For every pixel measurement we generate path samples by combining arbitrarily long eye and light paths. Intuitively, this results in having more than one technique to sample a path of a certain length, so the multiple importance sampling is employed to weight individual contributions.

As $\overline{x}_{s,t}$ we denote a path created by connecting a light path with $s$ vertices to an eye path with $t$ vertices. That said, the bidirectional estimator of pixel measurement $I$ is

$$\langle I \rangle = \sum_{s=0}^{\infty} \sum_{t=0}^{\infty} w_{s,t}(\overline{x}_{s,t}) \frac{f(\overline{x}_{s,t})}{p_{s,t}(\overline{x}_{s,t})} \ , \tag{2.55}$$

where $p_{s,t}(\overline{x}_{s,t})$ is the probability density of generating the path $\overline{x}_{s,t}$ using $s,t$ technique and $w_{s,t}(\overline{x}_{s,t})$ is the MIS weight of the path $\overline{x}_{s,t}$ generated by $s,t$ technique.

We deal with the infinite number of dimensions of the path space pretty much the same way as we did it in the case of path tracing. For each pixel measurement we create a single light path $\overline{y}$ and a single eye path $\overline{z}$ (see figure 2.12). Both paths are generated using random walk through the scene which is possibly terminated by sampling the zero contribution vertex $\epsilon$. We then create a various correlated path space samples by connecting each prefix of $\overline{z}$ to

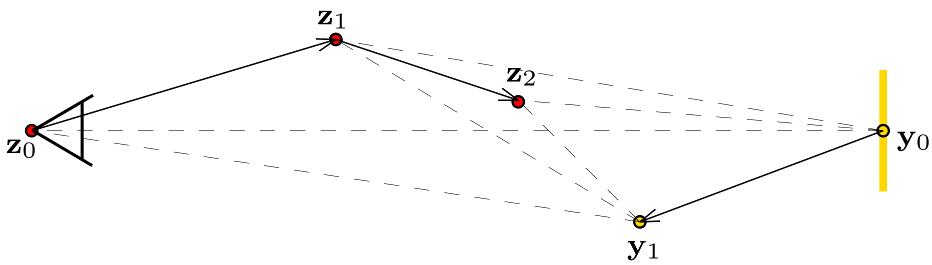


Figure 2.12: A single measurement during the bidirectional path tracing. A light path with 2 vertices is connected to an eye path with 3 vertices. Dashed lines depict the explicit subpath connections.

each prefix of $\overline{y}$, so we can write the estimator 2.55 using $\overline{y}, \overline{z}$ as

$$\langle I \rangle = \sum_{s=0}^{k(\overline{y})} \sum_{t=0}^{k(\overline{z})} w_{s,t}(\overline{y}_s \overline{z}_t) \frac{f(\overline{y}_s \overline{z}_t)}{p_{s,t}(\overline{y}_s \overline{z}_t)} . \tag{2.56}$$

Note that this way we create exactly one sample for each technique $s, t$ but we don't have to consider techniques $s > (k(\overline{y}) + 1)$ or $t > (k(\overline{z}) + 1)$, since their sample paths contain the zero contribution vertex. This was a very brief overview of the bidirectional path tracing. In the following text, mathematical formulation of the algorithm is given.

The estimator 2.56 can be rewritten to

$$\langle I \rangle = \sum_{s=0}^{k(\overline{y})} \sum_{t=0}^{k(\overline{z})} w_{s,t}(\overline{y}_s \overline{z}_t) \alpha_s^L(\overline{y}) C_{s,t}(\overline{y}, \overline{z}) \alpha_t^E(\overline{z}) , \tag{2.57}$$

where $\alpha_s^L$ and $\alpha_t^E$ depends entirely on the light path and on the eye path respectively. These two parameters can be evaluated recursively during a random walk as

$$\alpha_0^L(\overline{y}) = 1 \tag{2.58}$$
$$\alpha_1^L(\overline{y}) = \frac{T_L(\overline{y})}{p_A(\mathbf{y}_0)} \tag{2.59}$$
$$\alpha_s^L(\overline{y}) = \frac{f_s(\mathbf{y}_{s-3} \to \mathbf{y}_{s-2} \to \mathbf{y}_{s-1}) |\cos \theta_{\mathbf{y}_{s-2}\mathbf{y}_{s-1}}|}{p_\sigma(\mathbf{y}_{s-1})} \alpha_{s-1}^L \tag{2.60}$$

and

$$\alpha_0^E(\overline{z}) = 1 \tag{2.61}$$
$$\alpha_1^E(\overline{z}) = \frac{T_E(\overline{z})}{p_A(\mathbf{z}_0)} \tag{2.62}$$
$$\alpha_t^E(\overline{z}) = \frac{f_s(\mathbf{z}_{t-1} \to \mathbf{z}_{t-2} \to \mathbf{z}_{t-3}) |\cos \theta_{\mathbf{z}_{t-2}\mathbf{z}_{t-1}}|}{p_\sigma(\mathbf{z}_{t-1})} \alpha_{t-1}^E \tag{2.63}$$

The parameter $C_{s,t}(\overline{y}, \overline{z})$ is fully determined by paths being connected as

$$\begin{aligned} C_{s+t\leq 1}(\overline{y}, \overline{z}) &= 0 \\ C_{0,t}(\overline{y}, \overline{z}) &= L_e(\mathbf{z}_{t-1} \to \mathbf{z}_{t-2}) \\ C_{s,0}(\overline{y}, \overline{z}) &= W_e(\mathbf{y}_{s-1} \to \mathbf{y}_{s-2}) \\ C_{s,t}(\overline{y}, \overline{z}) &= f_s(\mathbf{y}_{s-2} \to \mathbf{y}_{s-1} \to \mathbf{z}_{t-1}) G(\mathbf{y}_{s-1}, \mathbf{z}_{t-1}) f_s(\mathbf{y}_{s-1} \to \mathbf{z}_{t-1} \to \mathbf{z}_{t-2}) \end{aligned} \tag{2.64}$$

As we can see, evaluating measurement contribution function is very straightforward and it can be done very efficiently. We just need to store parameters $\alpha_s^L$ and $\alpha_t^E$ at vertices during the process of a path generation. When we connect two subpaths $\overline{y}_s, \overline{z}_t$ we evaluate 2.64 and use the aforementioned parameters stored at last vertices of the paths to compute the unweighted contribution of the path $\overline{x}_{s,t}$.

**Algorithm 2** Bidirectional path tracing algorithm skeleton

```
function BidirectionalPathTracing(pixelSamples)
    for i ← 1 to pixelCount do
        image[i] = 0
        for j ← 1 to pixelSamples do
            pixelSample = GenerateRandomPixelSample(i)
            eyePath = GenerateEyePath(pixelSample)
            lightPath = GenerateLightPath
            for each eyeVertex in eyePath do
                for each lightVertex in lightPath do
                    C = ComputeUnweigtedContribution(eyeVertex, lightVertex)
                    W = ComputeMISWeight(eyeVertex, lightVertex)
                    image[i] += (W * C) / pixelSamples
                end for
            end for
        end for
    end for
    return image
end function
```

Computing MIS weight is a harder process. There are $s+t+1$ sampling techniques to generate a path $\overline{x}_{s,t}$. We can simply use $w(\overline{x}_{s,t}) = \frac{1}{s+t+1}$. Of course, that makes 2.56 an unbiased estimator of the pixel measurement $I$ but we end up with worst of all techniques. If we want to lower the variance, we have use the balance or the power heuristic [Vea98]. Here we use the balance heuristic to simplify the notations.

$$w_{s,t}(\overline{x}_{s,t}) = \frac{p_{s,t}(\overline{x}_{s,t})}{\sum_{i=0}^{s+t} p_{i,s+t-i}(\overline{x}_{s,t})} \tag{2.65}$$

In 2.65 we can see that the weight depends on probability densities with which all the other techniques would have generated the path. Computing weights is a tricky process and naive implementation can be very inefficient and numerically unstable. Veach proposed a universal and relatively efficient approach, which requires a single iteration across path vertices and which is numerically stable. Recently, Antwerpen proposed another very efficient approach which is suitable for GPU implementations [vA11], because it doesn't require the iteration across path vertices. On the other hand, Antwerpen's approach is slightly limited Probabilities can not depend on subpaths lengths.

The following algorithm briefly summarizes the algorithm described. Details have been left out, since we will see how the *Vertex Connection and Merging* can be modified to perform the bidirectional path tracing in later chapters.

## 2.3 Density estimation methods

This section is devoted to so called density estimation methods for global illumination. Particularly, we are going to discuss theory and limitations of original photon mapping algorithm and its derivatives. Again, following text is only a brief overview of important concepts, which one needs to be familiar with in order to understand complex global illumination algorithms. References to relevant books and papers are given wherever possible.

### 2.3.1 A short introduction to particle tracing

Particle tracing can be roughly described as a process of finding an approximation of equilibrium radiance distribution (functions $L_i, L_o$) in a scene, so that we are able to use it to perform a measurement of any kind locally without the need of recursive sampling. It is based on generating $N$ samples of radiance function within the scene. Particle is a synonym for such sample in this context. Every particle $(\alpha_j, \overrightarrow{\omega_j}, \mathbf{p}_j)$ is represented by its weight $\alpha_j$, incoming direction $\overrightarrow{\omega_j}$ and position $\mathbf{p}_j$. The weights of particles must obey some conditions, so that

$$\frac{1}{N}\sum_{j=1}^{N}\alpha_j W_e(\mathbf{p}_j, \overrightarrow{\omega_j}) \tag{2.66}$$

is an unbiased estimator of

$$\int_A \int_{\mathcal{S}^2} W_e(\mathbf{p}, \overrightarrow{\omega}) L_i(\mathbf{p}, \overrightarrow{\omega})\, d\sigma^{\perp}_{\mathbf{p}}(\overrightarrow{\omega}) dA(\mathbf{p})\ . \tag{2.67}$$

The main advantage of particle approach is that we can create our approximation of equilibrium radiance once and then use it to perform numerous measurement (e.g. all pixels). Of course, this results in correlation between measurements but that is often not objectionable.

### 2.3.2 Photon mapping

Photon mapping was originally formulated by Jensen. It is a quite complicated global illumination algorithm and here we discuss only a basic version that presents the idea behind it. Our formulation is in coherence with the original one, which means we talk about flux transport. Photon mapping can also be formulated using Veach's particle tracing framework introduced in the previous section, which is not that intuitive but more robust. Extensions to the basic version are shortly discussed at the end of this section.

The basic definition of the photon mapping consists of two main steps. At first, we generate the photon map by tracing photons through the scene. After that, photon map is used to produce biased estimate of all pixel measurements by tracing paths from the camera. All versions of algorithm mostly differs in what way is the photon map used. The algorithm 3 briefly outlines the basic photon mapping through the pseudocode.

**Algorithm 3** Basic photon mapping algorithm

```
function PhotonMapping
    photonMap = CreatePhotonMap
    for i ← 1 to pixelCount do
        pixelSample = GenerateRandomPixelSample(i)
        ray = MapPixelSampleToRay(pixelSample)
        hitPoint = TraceRay(ray)
        image[i] = PhotonMapRadianceEstimate(photonMap, ray, hitPoint)
    end for
    return image
end function
```

#### 2.3.2.1 Photon map generation

Photon map is generated by tracing $N$ particles (energy packets) through the scene from the light source. At every intersection of a particle with a non-specular surface, photon is stored to the photon map. Each photon is described by its weight (carried energy) $\alpha$, incident direction $\overrightarrow{\omega}$ and position $\mathbf{p}$. The figure 2.13 visualizes the process of photon map generation. It is good, if all photons stored in the photon map have roughly the same weight, which means that light intensity at some location is described also by the density of photons. Disadvantage of having photons with varying weights is that a single photon with a large weight might introduce artifacts to the rendered image. This will become obvious when we discus radiance estimation using photon map.

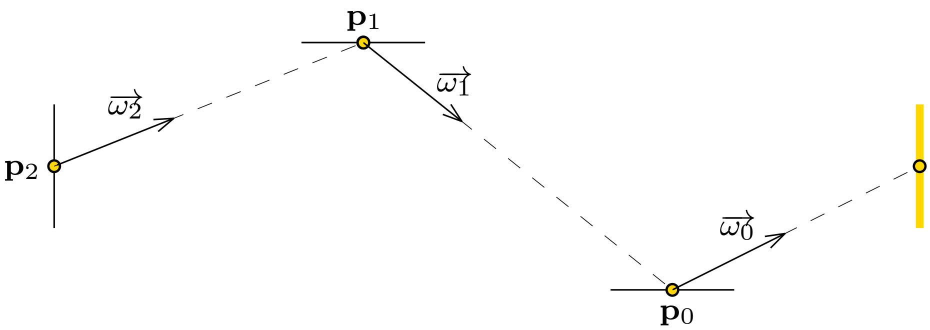


Figure 2.13: A single traced photon path during the process of photon map generation. The yellow line depicts the light source, which is the start point of every photon path. Individual photon records created by tracing depicted photon path are shown as yellow disks.

As mentioned, particles are traced from the light source and their initial position and direction is sampled from the distribution proportional to the emitted radiance function of the scene. Let us denoted the state of a particle at $i$-th vertex of its path as $(\alpha_i, \overrightarrow{\omega_i}, \mathbf{p}_i)$. The initial weight $\alpha_1$ is

$$\alpha_1 = \frac{|\mathbf{N}_{\mathbf{p}_1} \cdot \overrightarrow{\omega_1}| L_e(\mathbf{p}_1, \overrightarrow{\omega_1})}{p(\mathbf{p}_1, \overrightarrow{\omega_1})} , \tag{2.68}$$

where $p(\mathbf{p}_1, \overrightarrow{\omega_1})$ is the probability density of the ray $(\mathbf{p}_1, \overrightarrow{\omega_1})$ with respect to the throughput measure.

Particle is then stochastically traced through the scene and its state is stored to the photon map if it intersects a non-specular surface. The weight of the stored photon at $i$-th path vertex is equal to the weight of the incident particle, $\alpha_{i-1}$, and the incident direction of the photon is the opposite to the incident particle direction, $-\overrightarrow{\omega}_{i-1}$. Tracing is possibly terminated by Russian roulette. Particle weight is modified upon scattering at $i$-th path vertex $(i \geq 2)$ as

$$\begin{aligned} \alpha_i^* &= \alpha_{i-1} \tfrac{f_s^*(\mathbf{p}_i, -\overrightarrow{\omega}_{i-1}, \overrightarrow{\omega_i}) |\mathbf{N}_{\mathbf{p}_i} \cdot \overrightarrow{\omega_i}|}{p_\sigma(\overrightarrow{\omega_i})} \\ \alpha_i &= \tfrac{1}{q_i} \alpha_i^* \, , \end{aligned} \tag{2.69}$$

where $\overrightarrow{\omega_i}$ is the new direction of the particle generated with the probability density proportional to the adjoint BSDF defined against the solid angle measure. The coefficient $q_i$ is the discrete probability of continuing the path of our particle. Recall that we want all photons to have approximately the same weight. That means we set it to

$$q_i = \frac{\alpha_i^*}{\alpha_{i-1}} \, . \tag{2.70}$$

The number of photons stored in the photon map is usually (e.g. for closed environment) much larger than the initial number of traced particles $N$. The number of photons required is mostly given in advance and particles are traced until the specified amount is reached. Usually, hundreds of thousands or millions of photons are stored. The more photons, the better approximation of equilibrium radiance within the scene. We are only limited by the size of the memory.

#### 2.3.2.2 Rendering using photon map

Measurement estimation using photon map is relatively easy in its basic form, which consist in direct visualization of the map. At first, ray is traced from the camera through each pixel until it hits non-specular surface (e.g. recursive ray tracing). At such intersection $(\mathbf{p}', \overrightarrow{\omega}')$, outgoing radiance $L_o(\mathbf{p}', \overrightarrow{\omega}')$ is measured using $\langle L_o, L_i \rangle$. The figure 2.14 visualizes the process of rendering from a photon map.

The problem here is that we would like to have *enough* photons (incident radiance samples) at the differential surface around point $\mathbf{x}'$. The probability density of having even one such photon is almost zero if we trace only a finite number of photons, therefore we interpolate $L_i$ from nearby photons using density estimation. This is the point where the bias is introduced to the measurement. The standard density methods used in the photon mapping is k-nearest neighbors. The estimate of the $L_o(\mathbf{p}', \overrightarrow{\omega}')$ is then

$$L_o(\mathbf{p}', \overrightarrow{\omega}') \approx \frac{1}{N\pi r^2} \sum_j^n \alpha_j f_s(\mathbf{x}', \overrightarrow{\omega}', \overrightarrow{\omega_j}) \, , \tag{2.71}$$

where $r$ is the minimal radius containing $n$ nearest neighbors or some predefined maximal value. This is a simplest methods possible. It would be better to use some kernel density

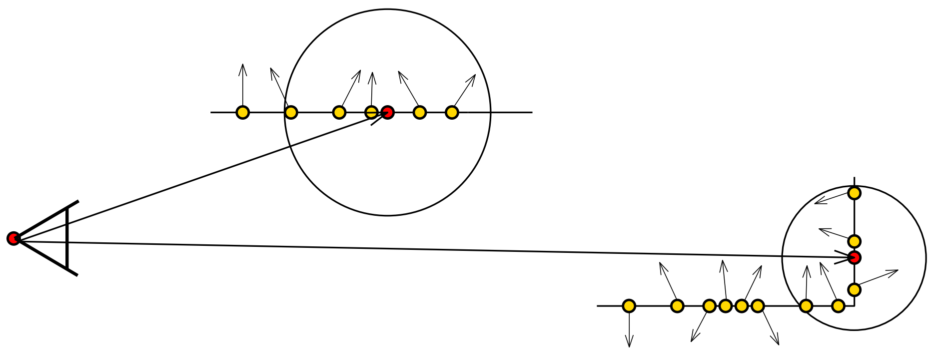

Figure 2.14: Rendering from the photon map. Figure depicts the process of radiance estimation for two pixels. Red discs represent hitpoints created by tracing primary rays from the camera. Yellow discs represent photons stored in the photon map. A single thin arrow points in the direction from which is a particular photon incident. The circles visualizes the photon search radius. Note that we required 5 photons for a single radiance estimate.

estimation. Note that by increasing the number of photons in the photon map and the number of photons considered around the point $x'$, we get more accurate approximation to the outgoing radiance. In fact, when $N \to \infty$ we are able to acquire the exact value of $L_o(\mathbf{x}', \overrightarrow{\omega}')$ with the cost of infinite amount of memory. That is why the method is said to be consistent.

#### 2.3.2.3 Extensions

Previous text described the basic concepts behind the photon mapping. Anyway, rendering method presented is mostly all but useful, because it results in an image containing low frequency noise even if large number of photons is stored, especially in the scenes with glossy materials. For this reason, direct illumination is usually estimated like in the standard path tracing. Photon map is only used when estimating indirect illumination. Nevertheless, photon map is often good for handling directly visible caustics. For this reason a separate photon map (caustic map) is created during the process of photon tracing, which is then used to estimate the illumination caused by caustics.

### 2.3.3 Progressive photon mapping

Recall that photon mapping is a consistent algorithm but in order to reach the exact solution it requires an inifinite number of particles to be traced and an infinite number of photons to be searched around fixed point $\mathbf{x}'$, where we want to estimate the outgoing radiance $L(\mathbf{x}', \omega')$. More precisely

$$L_o(\mathbf{x}', \omega') = \lim_{N \to \infty} \frac{1}{N \pi r^2} \sum_{j}^{N^\beta} \alpha_j f_s(\mathbf{x}', \overrightarrow{\omega}', \overrightarrow{\omega_j}) \tag{2.72}$$

where $\beta \in (0, 1)$, so that $N^\beta$ is also an infinite value infinitely smaller than $N$ in the limit. Note that $r$ in 2.72 converges to $dA(x')$. Solving the previous equation, of course, requires an infinite amount of memory, when done using the original photon mapping. Progressive photon mapping (PPM) [HOJ08] does it without the infinite memory requirement. It is a multipass algorithm. The first pass consist in standard recursive ray tracing for each pixel. Ray tracing is always terminated at first non-specular vertex and such vertex is stored to some array together with the required data for the radiance estimation. Every consecutive pass is standard photon tracing. A new photon map is build during it, which is used to refine the radiance estimates for all stored hitpoints from the first pass. The following algorithm outlines the progressive photon mapping through the pseudocode.

**Algorithm 4** Progressive photon mapping algorithm

```
function ProgressivePhotonMapping(numOfPasses)
    for i ← 1 to pixelCount do
        pixelSample = GenerateRandomPixelSample(i)
        ray = MapPixelSampleToRay(pixelSample)
        hitPoint = FirstNonSpecularHit(ray)
        pixelHitPoints += hitPoint
    end for
    for i ← 1 to numOfPasses do
        photonMap = CreatePhotonMap
        RefineRadianceEstimate(photonMap, pixelHitPoints, image)
    end for
    return image
end function
```

At every stored hit point $\mathbf{x}'$ from the first pass we store the current density estimation radius, $R(\mathbf{x}')$, number of photons in the radius, $N(\mathbf{x}')$, and the accumulated unnormalized reflected weight

$$\tau(\mathbf{x}') = \sum_{j=1}^{N(\mathbf{x}')} \alpha_j f_s(\mathbf{x}', \vec{\omega}', \vec{\omega_j}) \; . \tag{2.73}$$

The initial radius $R(\mathbf{x}')$ is determined by searching certain amount of nearest photons after the first photon tracing pass.

### Radius reduction

We denote the number of photons that ends up in the search radius $R(\mathbf{x}')$ of the point $\mathbf{x}'$ after $i \geq 2$-th photon tracing pass as $M(\mathbf{x}')$. We want to reduce the radius $R(\mathbf{x}')$ but we also want to increase the precision of our radiance estimate by increasing the photon density $\frac{N(\mathbf{x}')}{\pi R(\mathbf{x}')^2}$. We do it by using a single parameter $\alpha \in (0, 1)$ which controls the amount of photons we keep after the new photon tracing path. We can write the new number of photons at point $\mathbf{x}'$ as

$$\hat{N}(\mathbf{x}') = N(\mathbf{x}') + \alpha M(\mathbf{x}') \; . \tag{2.74}$$

This leads to the new possibly reduced density estimation radius at $x'$

$$\hat{R}(\mathbf{x}') = R(\mathbf{x}')\sqrt{\frac{N(x') + \alpha M(\mathbf{x}')}{N(\mathbf{x}') + M(\mathbf{x}')}} \ . \tag{2.75}$$

For more elaborate derivation, we refer the interested reader to the original paper [HOJ08].

#### Weight correction

We also need to correct the $\tau(\mathbf{x}')$ at the hit point $\mathbf{x}'$ for the reduced radius. The accumulated reflected weight of $M(\mathbf{x}')$ number of photons in the old search radius $R(\mathbf{x}')$ is

$$\tau_M(\mathbf{x}') = \sum_{j=1}^{M(\mathbf{x}')} \alpha_j f_s(\mathbf{x}', \overrightarrow{\omega}', \overrightarrow{\omega_j}) \ . \tag{2.76}$$

Assuming that the weight density is constant in the search radius, we can write the accumulated reflected flux for the reduced radius as

$$\hat{\tau}(\mathbf{x}') = (\tau(\mathbf{x}') + \tau_M(\mathbf{x}'))\frac{\pi \hat{R}(\mathbf{x}')}{\pi R(\mathbf{x}')} \tag{2.77}$$

$$= (\tau(\mathbf{x}') + \tau_M(\mathbf{x}'))\frac{N(\mathbf{x}') + \alpha M(\mathbf{x}')}{N(\mathbf{x}') + M(\mathbf{x}')} \tag{2.78}$$

Again, we refer to more elaborate derivation provided in the original paper [HOJ08].

#### Radiance evaluation

After every photon tracing pass, we present the new refined image just by iterating through stored hit points from the first pass and computing the radiance estimate from the data stored. For a particular hitpoint $x'$, the estimate is evaluated as

$$L_o(\mathbf{x}', \overrightarrow{\omega}') \approx= \frac{\tau(\mathbf{x}')}{\pi R(\mathbf{x}') N_{emitted}} \tag{2.79}$$

where $N_{emitted}$ is the total number of particle traced acquired by summing the number of particles traced from all tracing passes done so far.

#### Discussion

PPM does not solve many problems of previously presented version of photon mapping, because its convergence is still slow and even by tracing tens of millions of particles, we will not get a clean result when glossy materials are present in the scene. On the other hand, this algorithm is suited for the scenes with specular and diffuse materials. Also, it is relatively easy to extend the photon mapping algorithm presented to this progressive version.

### 2.3.4 Stochastic progressive photon mapping

Stochastic progressive photon mapping (SPPM) [HJ09] is the extension of the progressive photon mapping which tries to solve practical limitations of the original algorithm. It is capable of capturing distributed ray tracing [CPC84] effects (e.g. glossy reflections) in a reasonable amount of time. We will not derive the algorithm here. Interested reader can always read the original work by Hachisuka [HJ09]. Basically, the SPPM is almost the same as PPM. The change is that the stored hit points, generated by tracing paths from the camera, are regenerated after every photon tracing pass using distributed ray tracing.

### 2.3.5 Bidirectional photon mapping

Vorba [Vor11] recently formulated the photon mapping using the path integral framework[Vea98] which allowed him to formulate an algorithm that performs radiance estimate at different vertices of an eye path using the photon map and combine the results using the multiple importance sampling. The algorithm is much alike bidirectional path tracing, except that light paths are traced for the purpose of creating a photon map only and they are not connected to eye paths. Naturally, the author calls his algorithm bidirectional photon mapping. The algorithm itself is not very practical because of its slow convergence. On the other hand, formulating the photon mapping in the path integral framework was a step toward successful combination of path sampling methods with density estimation methods as we will see in the next chapter.

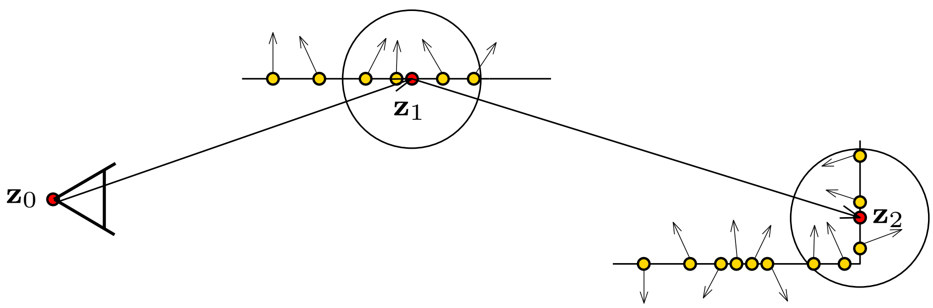


Figure 2.15: Visualization of radiance estimate for a single pixel within bidirectional photon mapping. An eye path is traced through the scene. Photon map radiance estimate is performed at each eye vertex (red disc) and the results are combined using the multiple importance sampling

# Chapter 3

# Vertex connection and merging

Both presented families of global illumination algorithms, the path sampling and the density estimation, have their own strengths and weaknesses. Path sampling algorithms, especially bidirectional path tracer, are good at sampling almost all light paths. The weakness of this class is the computational complexity. Recall that to evaluate bidirectional path sample contribution we had to perform a visibility check between endpoints of light and eye paths, which is relatively expensive operation. Density estimation algorithms (e.g. photon mapping) mostly have worse mean squared error convergence rate than path sampling algorithms. Their main advantage is that we can have large number of samples and account for their contributions by a single range query. Density estimation methods are also good when it comes to sampling so called $SDS$ paths, which often have small probability density in the context of path sampling [Vea98]. In this chapter we present the theory behind the vertex connection and merging algorithm (VCM) [GKDS12]. The idea behind it is to combine the bidirectional path tracing (BPT) with the photon mapping (PM) into a single robust algorithm.

From the success of multiple importance sampling in the bidirectional path tracing, one can judge, that this Monte Carlo technique can be used to successfully combine BPT and PM. Recall that MIS estimator of an integral $I = \int_\Omega f(\overline{x}) d\mu_A(\overline{x})$ is

$$\langle I \rangle = \sum_{i=1}^{m} \frac{1}{n_i} \sum_{j=1}^{n_i} w_i(x_{i,j}) \frac{f(x_{i,j})}{p_i(x_{i,j})} \,, \tag{3.1}$$

where $m$ is the number of sampling techniques, $n_i$ is the number of samples taken from $i$-th technique, $x_{i,j}$ is $j$-th sample taken by the $i$-th sampling technique, $w_i$ is the weighting function of $i$-th technique and $p_i$ is the probability density with respect to the measure $\mu$ of $i$-th technique. Note that to use the MIS estimator, all sampling densities have to be defined against the same measure. For this reason we need to express both algorithms in the same mathematical framework. We choose the path integral framework, because bidirectional path tracing with multiple importance sampling is already successfully formulated.

## 3.1 Vertex connection

Consider a path $\overline{x}$ of the fixed length $k$. Recall that in bidirectional path tracing this path can be sampled by $k+2$ techniques by gradually setting the $s = 0, 1, \ldots, k+1$, where $s$ denotes the number of vertices sampled from the light source. We will refer to the bidirectional sampling techniques as *vertex connection*, because we need to perform an explicit visibility check to evaluate the contribution of a path, except in case of $s \in \{0, k+1\}$, which corresponds to the unidirectional path tracing and we just need to evaluate the emitted radiance function. A path sampled by any vertex connection technique will be called *regular path*. We denote the probability density of sampling the regular path $\overline{x}$ using $s$ light path vertices and $t$ eye path vertices as $p_{VC,s,t}$. Figure 3.1 depicts a single path sampled by a vertex connection technique.

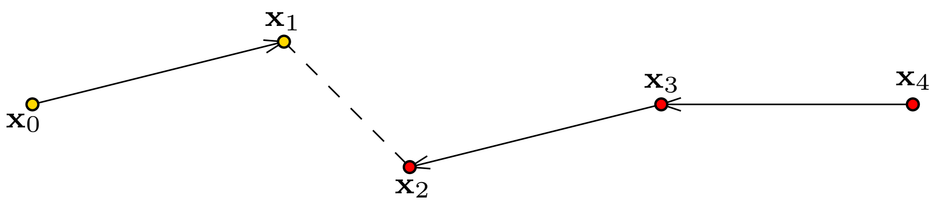


Figure 3.1: A path of length $k = 4$ sampled by the vertex connection technique $s = 2\,, t = 3$.

## 3.2 Vertex merging

Consider, we are performing the radiance estimate using a photon $\mathbf{x}_s^*$. The tracing history of our photon $(\mathbf{x}_0, \ldots, \mathbf{x}_{s-1})$ and the current eye path $(\mathbf{x}_s, \ldots, \mathbf{x}_k)$ together form a regular path $\overline{x}$ of the length $k$. The photon can be used to approximate the measurement contribution of $\overline{x}$ . We will refer to the photon mapping techniques for sampling regular paths as *vertex merging*, because we can intuitively imagine it as merging vertices $\mathbf{x}_s^*$ and $\mathbf{x}_s$. Figure 3.2 depicts sampling of the path from figure 3.1 by vertex merging at the last eye path vertex.

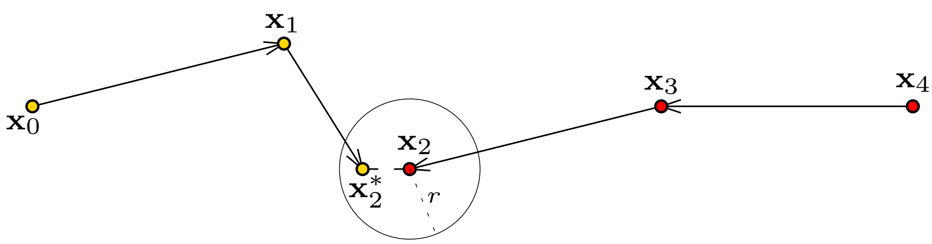


Figure 3.2: A path of length $k = 4$ sampled by the vertex merging technique $s = 3\,, t = 3$.

We need to find the probability density of our regular path $\overline{x}$ proposed by the vertex merging at $\mathbf{x}_s^*$. Note that our path $\overline{x}$ could have been proposed by any photon located within the merging radius $r$ at the endpoint of the eye path. The discrete probability density of proposing the regular path through vertex merging is then

$$P_{acc}(\overline{x}) = Pr(\|\mathbf{x_s} - \mathbf{x_s^*}\| < r) = \int_{A_d} p_A(\mathbf{x}_{s-1} \to \mathbf{x}) dA(\mathbf{x}) \,, \tag{3.2}$$

where $A_d = \{\mathbf{x} \in \mathcal{M} \mid \|\mathbf{x_s} - \mathbf{x}\| < r\}$. If we accept that probability density of having a photon in our merging radius is constant, we can approximate $P_{acc}$ as

$$P_{acc}(\overline{x}) \approx \pi r^2 p_A(\mathbf{x}_{s-1} \to \mathbf{x}_s^*) \ . \tag{3.3}$$

Note that the probability $P_{acc}(\overline{x})$ is dimensionless, so the probability density of proposing the regular path $\overline{x}$ through vertex merging is approximately

$$p_{VM,s,t}(\overline{x}) \approx p_{VC,s,t}(\overline{x}) P_{acc}(\overline{x}) \ . \tag{3.4}$$

Note that $p_{VM}$ is defined against the same product area measure as the probability density of any vertex connection technique for $\overline{x}$ . The photon serves as a random variable that conditions the acceptance of our regular path.

From the definition 3.4, we can see that the vertex merging probability density of sampling a regular path is at most same as the probability density of sampling the same path using the vertex connection technique. In fact, for practical sizes of merging radius, the density of vertex merging will be units of magnitude lower than the density of vertex connection. The real power of vertex merging techniques lies in relatively low computational cost. We can account for tens of millions of samples in the same time we would evaluate a single vertex connection sample, because, instead of visibility check, we just have to perform range search. We can think of vertex merging as a kind of brute force variance reduction technique.

We will approximate the measurement contribution of a regular path $\overline{x}$ , proposed by a vertex merging technique, using the function $f_{VM}$ defined as

$$f_{VM}(\overline{x}) = T_L(\mathbf{x}_0, \dots, \mathbf{x}_{s-1}) \, \frac{1}{\pi r^2} \, f_s(\mathbf{y}_{s-1} \to \mathbf{z}_{t-1} \to \mathbf{z}_{t-2}) T_E(\mathbf{x}_s, \dots, \mathbf{x}_k) \ , \tag{3.5}$$

where $r$ is the merging radius at the vertex $\mathbf{x}_s$. Note that we can replace the $\frac{1}{\pi r^2}$ term in 3.5 by a density estimation kernel $K_r(\|\mathbf{x_s^*} - \mathbf{x_s}\|)$ with the support radius $r$. $\frac{1}{\pi r^2}$ is used, because it is coherent with our definition of photon mapping.

In further discussion, if we put an extended path $\overline{x}^*$ into a probability density function $p_{VC}$, we mean the probability density of the regular path obtainable from $\overline{x}^*$ by leaving out the photon vertex $\mathbf{x}_s^*$. Putting a regular path into a probability density function $p_{VM}$ is analogous to the previous case.

## 3.3 Measurement estimation

Now that we have formulated the photon mapping as a set of techniques for sampling regular path using the path integral framework, we can combine it with the bidirectional path tracing using multiple importance sampling. The result is a single estimator for a pixel measurement $I$ defined as

$$\begin{aligned} \langle I \rangle_{VCM} = \frac{1}{n_{VC}} \sum_{l=1}^{n_{vc}} \sum_{s=0}^{\infty} \sum_{t=0}^{\infty} w_{VC,s,t}(\overline{x}_{s,t}) \frac{f_{VC}(\overline{x}_{s,t})}{p_{VC,s,t}(\overline{x}_{s,t})} \quad + \\ \frac{1}{n_{VM}} \sum_{l=1}^{n_{vm}} \sum_{s=2}^{\infty} \sum_{t=2}^{\infty} w_{VM,s,t}(\overline{x}_{s,t}) \frac{f_{VM}(\overline{x}_{s,t})}{p_{VM,s,t}(\overline{x}_{s,t})} \end{aligned} \quad , \tag{3.6}$$

where $n_{VC}$ is the number of bidirectional samples, $n_{VM}$ is the number of traced particles. Note that this definition does not consider the vertex merging at the camera lens ($t = 1$). Usually, $n_{VM} = 1$ and $n_{VC} \geq$ 1mil. When power heuristic is used, the weighting functions can be computed as

$$w_{v,s,t}(\overline{x}_{s,t}) = \frac{n_v^\beta p_{v,s,t}^\beta(\overline{x}_{s,t})}{n_{VC}^\beta \sum\limits_{s' \geq 0, t' \geq 0} p_{VC,s',t'}^\beta(\overline{x}_{s,t}) + n_{VM}^\beta \sum\limits_{s' \geq 2, t' \geq 2} p_{VM,s',t'}^\beta(\overline{x}_{s,t})}, \tag{3.7}$$

where $v \in \{VC, VM\}$. The approach to the weight evaluation is much more crucial in VCM than it is in bidirectional path tracing, because of commonly large number of vertex merging samples.

## 3.4 The VCM algorithm

This section gives a brief outline of the original VCM algorithm [GKDS12]. Detailed description together with an implementation is provided in the next chapter.

The basic version of the algorithm consist of two main phases, particle tracing and image measurement. The goal of particle tracing step is to generate a standard photon map. This is similar to the presented photon mapping algorithm. The difference is that particle paths are also saved and their number corresponds to the number of image pixels. The process of image measurement is really a combination of bidirectional path tracing and bidirectional photon mapping [Vor11]. An eye path is sampled for each pixel measurement, possibly terminated by the Russian roulette. Every eye subpath of length $\geq 1$ is merged with all photons in the search radius, which is global for all pixels within a single iteration. The corresponding stored light subpath is used to evaluate the bidirectional estimate. The algorithm 5 outlines the vertex connection and merging version presented in the original paper [GKDS12]. Details are intentionally left out since they are described in the next chapter, when we discuss the implementation of VCM.

### Achieving consistency

Algorithm given above is inconsistent for any finite size of the merging radius. Images generated by it contains systematic error due to density estimation. We can make the algorithm consistent by running it in a loop with decreased merging radius every iteration. Then if we average results from every iteration, we get an image without systematic error as the number of iterations approaches infinity. The radius reduction scheme proposed by the original paper [GKDS12] is

$$r_i = r_1 \sqrt{i^{\alpha - 1}}, \tag{3.8}$$

where $\alpha \in (0, 1)$ is a user parameter and $r_1$ is the initial radius. These two values are discussed together with the implementation in the next chapter.

**Algorithm 5** Vertex connection and merging

```
function VertexConnectionAndMerging(numOfIterations)
    for i ← 1 to numberOfIterations do
        for j ← 1 to numOfPixels do
            lightsPaths[j] = GenerateLightPath
            photons += lightPaths[j]
        end for
        accel = BuildRangeSearchAccelerator(photons)
        for j ← 1 to numOfPixels do
            radiance = 0
            pixelSample = GenerateRandomPixelSample(i)
            eyePath = GenerateEyePath(pixelSample)
            for each lightVertex in lightPaths[j] do        ▷ Light tracing
                radiance += ConnectToCamera(lightVertex)
            end for
            for each eyeVertex in eyePath do
                for each lightVertex in lightPaths[j] do
                    radiance += ConnectVertices(eyeVertex, lightVertex)
                end for
                radiance += MergeVertices(eyeVertex, accel)
            end for
            image[j] += radiance/numOfPixels
        end for
    end for
    return image
end function
```

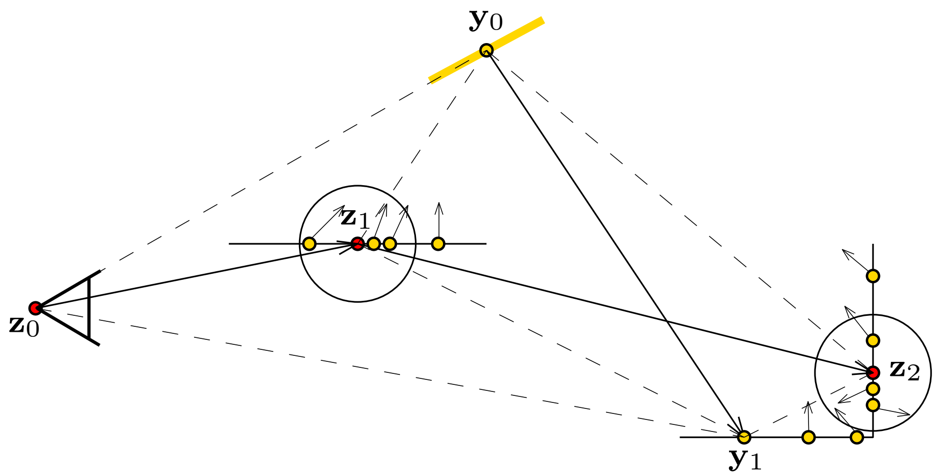


Figure 3.3: This figure depicts the vertex connection and merging estimate for a single pixel. Each eye path vertex (red disc) is connected to each light path vertex (yellow disc) as in bidirectional path tracing. Additionally, photon mapping radiance estimate is performed at each eye path vertex.

## 3.5 Convergence

The original paper shows that the progressive VCM algorithm has mean squared error convergence rate same as the BPT, which is $O(1/N)$ [Vea98]. As the number of iterations approaches infinity, the result contains vertex connection contributions only, because MIS weights for vertex merging contributions are zero for infinitely small merging radius. The gain from vertex merging is that we can have visually plausible result after a lower amount of samples than we would require with bidirectional path tracing. For more elaborate discussion on the convergence, we refer the interested reader to the original work [GKDS12].

# Chapter 4

# Implementation

The goal of this thesis was to analyze complex methods for global illumination through their implementations. We wanted to realize and compare a substantial subset of rendering algorithms. In order to ease the development process and accomplish a fair comparison of all implemented algorithms there was a demand for a common framework. All algorithms we wanted to realize were based on Monte Carlo integration and ray tracing, so we knew that very large amount of code will be shared among various rendering methods. We had an option to base our implementation on an existing rendering library, which is for example Pharr's *Physically Based Rendering Toolkit* [PH10], which has a large amount of capabilities, it contains implementation of numerous rendering algorithms, many material models, many camera models etc. The problem was, among others, the lack of interactivity. Beside the analysis of complex rendering methods, we wanted to bring Monte Carlo algorithms to interactivity through image filtering.

Although the focus of this thesis is primarily the discussion of complex global illumination algorithms, the first part of this chapter introduces the common framework, which served as the base for implementations of all required algorithms. We named it *Global Illumination Rendering Toolkit* (GIRT). We believe that understanding a generic rendering library is equally important as understanding global illumination in general. The second part is devoted to the description of the *vertex connection and merging* implementation. Recall that it is a robust algorithm that combines ideas from the path sampling and the density estimation, so understanding this renderer is equivalent to understanding bidirectional path tracing and progressive photon mapping separately.

## 4.1 Global Illumination Rendering Toolkit (GIRT)

This section presents the overall design of GIRT. It is intended to provide an overview of the library and we won't go into details. The discussion level should be deep enough for someone, who wants to start using the library, and it should be high enough for someone, who just wants to know, how a global illumination framework might be designed. Our discussion starts with the design goals and then it continues with particular components.

### 4.1.1 Design goals

We needed a library that allows us to implement various rendering algorithms while minimizing the code duplication. The first mandatory requirement on it was that it had to work within *Virtual Reality Universal Toolkit* (VRUT). As its name suggest, VRUT is a generic framework for development various applications related to the computer graphics. Its core provides a basic graphical user interface, framebuffer, and scene data. Users can then provide modules that work with the core scene data (e.g. render them). By implementing GIRT library we wanted VRUT to be able to present its internal data as realistically as possible.

Testing and analysis of various rendering algorithms within VRUT framework would be cumbersome, because the framework was not primarily designed to provide an environment for such an enterprise. That is why we needed our library to be independent of VRUT. It meant we needed a custom scene geometry representation, camera representation, threading support etc. It showed up that this would be necessary anyway, because VRUT was primarily designed for *real-time* rendering. That means it uses very different concepts in material library design and supports only delta distribution light sources.

The library had to be implemented using the C++ language. Not only because VRUT is implemented using this language but also because C++ is commonly used to develop this kind of applications for its robustness and speed. A clear design was the primary concern because we needed to allow rapid prototyping of rendering algorithms. That doesn't mean that efficiency of the implementation was of no concern to us but sometimes we traded it for well-arranged code. For example, we did not want to obfuscate our implementation by mixing CPU with GPU code. So these were the requirements more or less unrelated to actual rendering capabilities of the GIRT library.

**Scene geometry** Requirements around the scene geometry were relatively simple, because the library was supposed to create images of static scenes only (or simple dynamic). We just wanted to support various kinds of geometrical primitives and be able to add new ones in the future.

**Ray casting acceleration** As has been already mentioned, all algorithms we wished to implement are based on ray casting one way or another, so having a good acceleration structure that speeds up this operation is mandatory. Of course, we did not want to use the library to analyze various acceleration structures but it appeared to us that disallowing it by the design would be a pity, because it should not be too much work to account for it. That is why we decided to implement ray casting acceleration in a generic way.

**Light sources** A generic implementation of light sources to easily support various types of lights among all implemented rendering algorithms is a must for any robust physically based renderer, so we wanted GIRT to support as much types of light sources as possible and we did not want to account for different types explicitly within rendering algorithms implementations. Particularly, we wanted to support omindirectional light, environment light, directional light, spot light, and diffuse area light.

**Camera models** For the purposes of rendering algorithms analysis a single pinhole camera model would suffice. Nevertheless, there were ideas to use the GIRT library for the generation of HDR environment maps, so we had to create a generic camera model support.

**Materials** Supporting various surface material types is crucial if one wants to thoroughly test a rendering algorithm. We knew that we need to pay extra attention to the design around the material support if we didn't want to end up duplicating the code and also because we didn't want to implement *all* material types immediately. From the start we needed to support diffuse material, glossy material, and glass. A basic support of first to mentioned types can be achieved by exploiting the *phong shading model*[Pho75].

**Rendering and rendering algorithms** Rendering algorithms can be thought as the hearth of the library. We wanted the implementation of a new algorithm to be as smooth as possible and we wanted the result to be short and clean to minimize the risk of introducing an error. Every rendering algorithm has to run in parallel (e.g. custom thread), because we want to use the library in interactive applications.

From the start we wanted to support judiciously optimized path tracing. This algorithm is, in comparison to other methods like VCM, relatively easy to implement without making a mistake. The images generated by this algorithm can be then used as reference images when debugging complex algorithms.

**Image filtering** Aside performing an analysis of various rendering methods, we wanted to push complex global illumination algorithms towards interactivity through image filtering. We wished to test recently proposed efficient noise reduction algorithms like *Guided image filtering*[HST13]. We also wanted to be able to filter different types of image contributions(e.g. direct and indirect lighting) by different types of image filters, because different contributions sometimes exhibit different noise type.

### 4.1.2 Library overview

This chapter provides a high level overview of the GIRT library design and architecture. We discuss how we have fulfilled our design goals but we will not go into implementation details.

#### 4.1.2.1 Scene

If we take a look at the figure 4.1 we can see that an instance of the class **SceneCache** is responsible for holding geometry data and light sources of a single virtual scene. It does not contain information about a camera. These are separated, because we might want to view a single scene by different cameras and we might want to reuse a single camera among multiple scenes. More on cameras will be discussed later. An instance of the **SceneCache** also does not contain any material related data. This design decision was made, because we wanted to be able to keep multiple **SceneCache** instances in the memory and share material data between them. As we will discuss, a material can contain high-resolution textures together with their mipmaps, so a material can consume a large amount of memory.

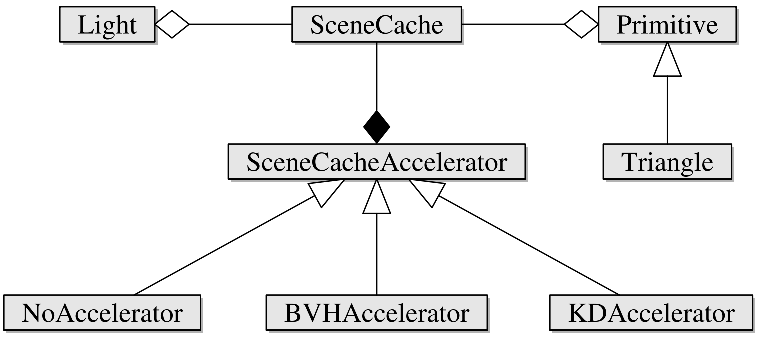


Figure 4.1: The architecture around the class **SceneCache**.

All scene data, geometry and light sources, have to be created through the interface of **SceneCache**. This is not just to underline that an instance of **SceneCache** owns this data but also to ensure that scene data are allocated efficiently without fragmenting the memory. Imagine a large scene consisting of millions of geometric primitives. Allocating these primitives using the standard allocator is time consuming and can eventually consume unnecessarily large amount of memory.

**SceneCache** also provides the interface for tracing rays through the scene geometry. It has two methods, one for casting a single ray and one for casting an arbitrarily large batch of rays. Casting in batches is more effective, because it delegates all requests to an instance of abstract class **SceneCacheAccelerator**, so there is a virtual call overhead. Nevertheless, batching is sometimes undesirable, because it may be cumbersome to employ it in implementation of specific rendering algorithm. As has been mentioned, an instance of **SceneCacheAccelerator** owned by the **SceneCache** instance is responsible for actual ray casting. Today, there are two types of ray casting acceleration structures implemented, kD-Tree (**KDAccelerator**) and cache optimized bounding volume hierarchy (**BVHAccelerator**). The type of the accelerator that an instance of **SceneCache** should use is specified using a fixed enumeration, which implies that new types can be added by the library developer only.

As has been stated earlier, GIRT is limited to static scenes only. This is quite a limitation that on the other hand allows certain simplifications while implementing ray casting acceleration structures. The process of feeding and instance of **SceneCache** with data has to be bounded by the calls to special **SceneCache** methods, so that the particular instance knows when to free the old data and when to build the acceleration structure for the new data. After the data is loaded, no change to the geometry is allowed. This might change in the near future.

### Primitive

The basic unit of the scene geometry is the abstract class **Primitive**. Recall our restriction on scene surface set $\mathcal{M}$. An instance of **Primitive** represents a single piece-wise differentiable 2-manifold. It can actually represent a set of such 2-manifolds but no current implementation does that. Discussing this, GIRT currently contains only one implementation of **Primitive** that represents a single triangle (class **Triangle**). There are certain parts of **Primitive** interface and responsibilities that we should discuss.

Every implementation of the **Primitive** class has to implement two ray intersection methods. The first method is used when we are performing an occlusion query (e.g. shadow ray) and the second one is used when we are performing the standard ray casting and searching for the first intersection (e.g. primary/secondary rays). The first method is usually easier to implement as well as it is usually faster than the second one, because we do not require information about surface for shading, we just want to know whether or not is a particular ray occluded.

Information about hit surface from primitive-ray intersection routine are recorded in an aggregate class **IntersectionInfo**. A single instance of this class saves the intersected primitive, distance to intersection, particular intersection point, normal vector associated with the intersection point, and other useful data required for shading (e.g. differentials for texture filtering).

Instance of **Primitive** can be queried for *world space* axis aligned bounding box of surface points it represents. This routine is particularly important for ray casting accelerators, because it is the only information they should know about the geometry of a primitive. Last but not least, a primitive can be a part of an area light source. For this reason **Primitive** provides a routine that allows uniform sampling of its surface points.

### Light

The class hierarchy regarding lights is depicted on figure 4.2. The base class **Light** was designed in the way, so that it is the only class rendering algorithms needs to be aware of. Through its interface a single light can be disabled or enabled, we can query for the total emitted power $[W]$, we can change the intensity, and we can query whether or not is a particular light source a delta distribution (e.g. point light). Delta distribution light sources require special care in renderers as they can not be sampled by following a random walk through BSDF sampling. This was the basic part of the **Light** interface. The other part, relatively complex, is related to the Monte Carlo sampling of a light surface points and directions. We refer the interested reader directly to the source source code of the library, because discussion of this interface is beyond the level of discussion in this section.

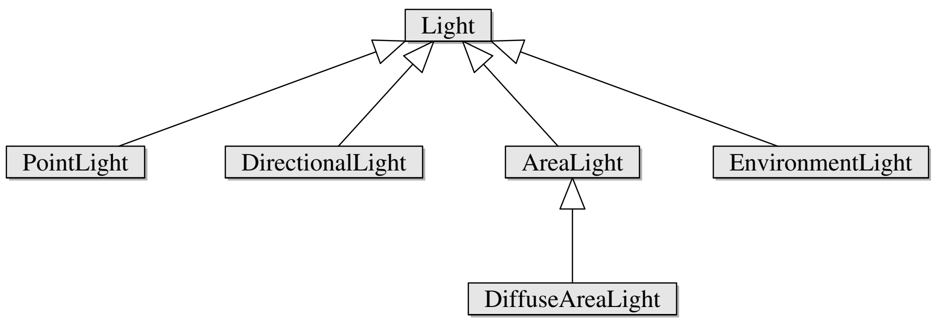


Figure 4.2: The architecture around the class **Light**.

Classes **PointLight** and **DirectionalLight** represent classic light sources known from the classic *OpenGL* fixed pipeline. Their names should suggest which particular light source each class represents. The direction of a directional light and the position of a point light can be changed even after the scene has been loaded and the rendering process started.

The class **EnvironmentLight** enables us to perform image based lighting. It is intended that there is at most a single light source of this type in a scene. An instance of **EnvironmentLight** is backed by a single latitude/longitude texture (possibly HDR).

**AreaLight** represents a generic interface to any type of area light source. There is currently only a single implementation that provides diffuse area light source (**DiffuseAreaLight**), which has constant $L_e$ from all points to all $\mathcal{H}^2_+(\mathbf{x})$ directions. Area light is assigned a portion of scene geometry (e.g. instances of **Primitive**), so there is no need to specially account for their geometry when performing *ray-scene* intersection. An instance of **Primitive**, which is returned from ray-primitive intersection routine in an instance of **IntersectionInfo**, contains, possibly non-null, reference to an instance of **AreaLight**.

#### 4.1.2.2 Materials

The Design of the material system is depicted on figure 4.3. An instance of **Primitive** holds a reference to a single instance of the **Material** class that describes its surface. Material itself is a hypothetically composed of various **BSDF** objects. We say "hypothetically", because, when asked, **Material** instance is capable of putting together an instance of **BSDF** that statistically describes the surface of the single point provided through instance of **IntersectionInfo**.

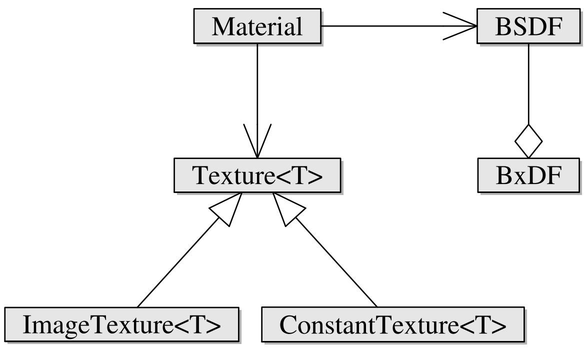


Figure 4.3: The architecture around the class **Material**.

#### BSDF

BSDF function for a specific surface point is encapsulated in the class with the same name, **BSDF**. Say, we have the result from a ray casting operation in the form of an **IntersectionInfo** instance and we want to perform the shading at the particular point. We ask the hit **Primitive** for its **Material**. Provided the instance of **IntersectionInfo**, the **Material** instance can give us the **BSDF** object for the hit point. The **BSDF** class is itself only an aggregate of **BxDF** class, which is an abstract class and every implementation should stand for a single BRDF or BTDF function(e.g. diffuse reflection). This design is similar to the one presented in Pharr's book on physically based rendering[PH10].

#### Textures

Consider the implementation of the diffuse (Lambertian) material, which is encapsulated in the class **MatteMaterial**. Recall that diffuse BRDF has only a single parameter, which tells how much of the incident energy is scattered. This parameter is provided to the specific **BxDF** object by the **MatteMaterial** instance through a texture lookup. The specific texture is polymorphically hidden behind an implementation of the abstract class **Texture**. Currently, there are two implementations. The class **ConstantTexture** stands for a singular specific value. This type of texture is fast to evaluate but it is not useful if we want to create a detailed scene without having too much geometry. The other type, **ImageTexture**, stores a rectangular array of values. The value for a specific **IntersectionInfo** is fetched from an instance of **ImageTexture** using *uv* coordinates. Image textures also support the mipmapping technique for filtering.

#### Supported materials

Beside the diffuse material, GIRT library currently supports phong material, mirror material, and glass material. All mentioned models are physically valid. Glass material is implemented in terms of the Fresnel equations. Physically plausible Phong material was implemented according to the technical report by Lafortune[LW94]. We intent to support many more material types in the future. The supported materials are enough to create realistically looking images, they were relatively easy to implement, and they are sufficient for analysis of various rendering algorithms.

As in the case of light sources, we did not talk about sampling support in the context of **BSDF**. For an high level overview, it should be sufficient to know that given an outgoing radiance/importance direction $\overrightarrow{\omega_o}$, an instance of **BSDF** provides an incident radiance/importance direction $\overrightarrow{\omega_i}$ sampled according to the probability density proportional to the reflectance function.

#### 4.1.2.3 Camera models

Take a look at the figure 4.4 which shows the design behind the camera support. All camera models are derived from a common base class, **Camera**. An instance of the **Camera** class holds the information about the *view* coordinate system of a camera (e.g. position and orientation). The **Camera** class provides large amount of methods in order to allow a comfortable change of the underlying view coordinate system. Currently, there are two subclasses of **Camera**. **PerspectiveCamera** implements standard pinhole camera model that is used to generate images and **SphericalCamera**, which is capable of creating HDR latitude/longitude environment maps.

#### 4.1.2.4 Rendering

The architecture around the rendering functionality of the GIRT library is described by the figure 4.5. The abstract class **AbstractRenderer** provides the rendering interface to the user as well as it offers common functionality to its subclasses. Every *final* subclass of **AbstractRenderer** should correspond to a single rendering algorithm (e.g. path tracing),

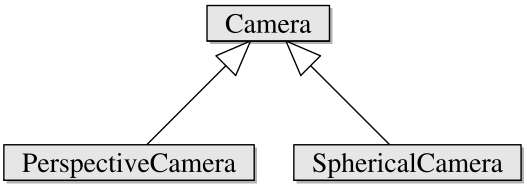


Figure 4.4: The architecture around the class **Camera**.

but it does not have to be that way. **AbstractRenderer** also owns the framebuffer, which can be customized by a particular implementation to fit its needs. Before anything can be rendered, an instance of **AbstractRenderer** has to be provided with a **SceneCache** instance and a **Camera** instance. The actual rendering process is controlled by two methods. They are so important that we even mention them here. Rendering is initiated by calling the function **startRendering()**. This method shall not hold the calling thread until the rendering is finished. It shall run the rendering in parallel and return as soon as possible. During the rendering process it is important to avoid changing any parameter that affects the process. For example, one should not change the intensity of a light source. The rendering process has to be stopped before changing any such parameter, which is done through the method **cancelRendering()**. This method shall hold the calling thread until the process is really terminated and it is safe to change a parameter affecting it.

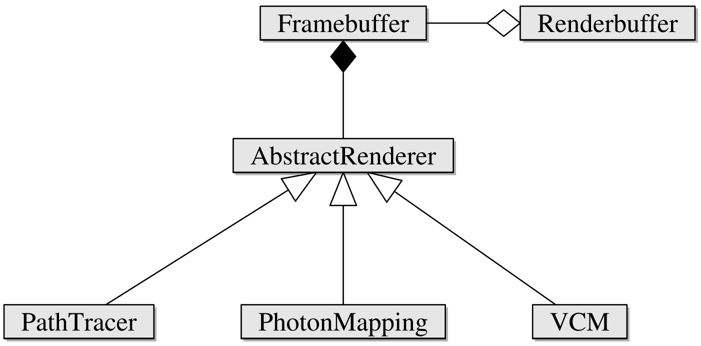


Figure 4.5: The architecture around the class **AbstractRenderer**.

Although we are going to discuss the **VCM** implementation in a detail within the second part of this chapter, we shortly discuss the **PathTracer** renderer here. This subclass implements the path tracing with the next event estimation and multiple importance sampling for direct lighting estimation. It is the simplest algorithm implemented within the GIRT library and it is used to create reference images whenever possible. Path tracing is based on the pure image sampling, which is why it is derived from the intermediate class **SamplerRenderer**. **SamplerRenderer** implements the **startRendering()** method of the **AbstractRenderer** base class by spawning a specific amount of worker threads. Each worker thread iteratively generates image samples and provides this image samples to the **PathTracer** for processing. In **PathTracer** implementation, every sample is mapped to a primary ray by using the camera, it is traced through the scene and its eventual contribution is recorded to the framebuffer.

Currently, there are three implementations of **AbstractRenderer**, the already discussed

**PathTracer**, **VCM**, and **PhotonMapping**. The last mentioned implements standard photon mapping as formulated by Jensen[Jen01] as well as stochastic progressive photon mapping[HJ09].

#### 4.1.2.5 Image filtering and tone mapping

The process of image filtering and tone mapping is completely handled by the *graphics processing unit* (GPU) and it is all encapsulated within a single class, **ImageFilter**. OpenGL API was used as the interface to the graphics card. Image filtering can be done on each component of a framebuffer separately. For example, **PathTracer** allows to separate contributions of direct lighting and indirect lighting. We can then leave the direct lighting component unfiltered and perform Gaussian filtering on the indirect lighting component. We have implemented three types of image filters, space-invariant Gaussian filter, bilateral filter[TM98], and guided image filter[HST13]. We refer the interested reader to the original papers for more information.

The results of image filtering are combined to a single texture, filtered image. The last step in the image pipeline is to perform the tone mapping, which allows to show the filtered image on today's *low dynamic range* display devices without loosing fidelity. Currently, only relatively simple global tone mapping operators are supported (e.g. Logarithmic, S-Curve, and Exponential).

#### 4.1.2.6 Implementation details

This small section is devoted to the discussion to some implementation details that are worth mentioning.

**Parallelization**

Global illumination algorithms that we have implemented (e.g. path tracing) are said to be *embarrassingly parallel*. For example, lock free parallelization in path tracing can be achieved just by relatively simple parallel reduction on image pixels, which is what the class **SamplerRenderer** exactly does. In the case of building photon map it is very much alike. For such tasks we have implemented the special class **AsynchronousTaskExecutor**. An instance of this class is capable of executing a given functor multiple times in parallel. We often set the number of parallel executions to the number of physical cores available on the machine and then we just adjust the thread priority of all executions.

Some tasks are not so easily made parallel. Building a hash grid among a set of point is just one example. It can be done using the parallel reduction and the prefix sum. For tasks like this we decided to use the Intel Thread Building Blocks[Int], which implements all common parallel algorithms in a generic way and its interface looks very much like the interface of C++ Standard Template Library (STL).

**Random numbers**

A correct implementation of a random number generation (RNG) is the first step when dealing with Monte Carlo methods. Testing the implementation before proceeding with implementation of an algorithm is worthwhile. By choosing a wrong RNG method, your algorithm may spend a lot of time on evaluating largely correlated samples. One needs an algorithm that really distributes its samples uniformly with sufficiently high resolution.

Another problem may arise if our stochastic algorithm is parallel, like in the case of rendering. If we seed individual instances of selected RNG in a wrong way, we can end up with each thread performing the same work. We have chosen our RNG according to guidelines provided in the article by David Jones[Jon].

### Color

The library is intended to produce visually plausible images of virtual scenes and not exact physical measurements. The light transport is computed independently only for three components of the *RGB* color model. Quantities for all wavelengths are packed in an instance of the class **RGBSpectrum**. The library is *hard coded* in the terms of this class, because we do not expect any changes of this decision in the future.

### Material and light definition files

We wanted the library to be a part of the *VRUT* framework as well as we wanted it to be standalone, but even VRUT provides us with a limited set of light types and material types. For these reasons we needed a proper way of defining the surface materials and scene lights. Using a 3rd-party format was out of the topic, because in the *VRUT mode* we still wanted to load all the geometry from its core and use most of its materials. We only wanted to be able to override its materials. For example, in GIRT we might have a better way of representing metal material than the simple Phong model. After thorough consideration we decided to create our own formats.

The structure of definition files is the same for both, materials and lights. Their syntax is very close to *JSON*, which is a data definition subset of *Javascript* language. Complete example files are provided together with the GIRT library. Appendix A and Appendix B show short example of material definition file and lights definition file respectively.

### Detailed geometry

To improve the detail of a scene we can either introduce more polygons or we can perturb geometric surface normals to change the shading behaviour. Each technique has its own strengths an weaknesses. Beside the large memory consumption and slower ray casting, having many small polygons can lead to numerical issues within the rendering algorithm. On the other hand, Veach [Vea98] showed that changing geometric surface normal can lead to various artifacts (e.g. light leaks or black spots). Naturally, GIRT supports the first approach, which consist in having more polygons. The second one is supported through the technique known as *bump mapping*. Bump map can be specified for any material (see appendix A).

Tree leaf is an example of something that requires large geometrical detail. When creating close up renders of such structure, we have no other choice than to introduce a large number of geometric primitives but when viewing it from a relatively large distance, we can represent it using a single planar polygon with texture map. GIRT supports *mask textures*, so a leaf can be represented by a single quadrilateral with a mask texture that says which part of the quadrilateral belongs to the leaf and which does not.

#### Numerical issues

The GIRT library is based on *IEE754 32bit* floating point numeric type, so numerical issues arise in many parts of the library. For example precision of ray casting is dependent on distance between the origin of a ray an the potential hit point. When tracing a ray from a location that was computed as the intersection of another ray with the scene we might have to offset the origin of the new ray in its direction to avoid an intersection of the surface around the location at a distance near zero. The required offset depends on the distance the previous ray traveled through the scene. An instance of **IntersectionInfo** records this information and provides the required offset for us.

## 4.2 Vertex connection and merging implementation

This section provides a discussion of the implementation of vertex connection and merging algorithm that exists withing the GIRT library. Unfortunately, there is not enough space to provide the complete source code here and outline it step by step. Instead, we first discuss the implementation in overall and then we provide the pseudocode of the main rendering method in the form of literate programming as it is used in the Pharr's book on physically based rendering [PH10]. The pseudocode matches the actual implementation as much as possible. The reader is assumed to be familiar with the basic concepts of the GIRT library and with the theory behind the vertex connection and merging method. We also recommend reading the technical report on implementing the VCM[Geo12] before continuing any further.

Our implementation more or less matches the algorithm definition in the original paper[GKDS12] and it uses recommendations provided in the technical report[Geo12]. For example the multiple importance sampling weights are computed using the modified Antwerpen's scheme[vA11]. This scheme is very efficient since its computation of the MIS weights has constant complexity as opposed to the linear complexity of the scheme proposed by Veach[Vea98] for bidirectional path tracing. However its efficiency comes at some cost. The probability densities of the sampled vertices can not depend on the length of a path. For example if we want to perform the Russian roulette not before a specific amount of vertices has been sampled, we have to disregard this in MIS weight computation and accept the bias or we have to use a different scheme, which may not be so efficient. Withing a bidirectional path tracing this is not an issue since there are a lot less weight computations than in the vertex connection and merging.

Our implementation supports all light types and materials that are implemented within the GIRT library, which means that besides diffuse BSDF and area lights it supports specular materials (e.g. mirror and glass) as well as delta distribution light sources and environment lights. Supporting all of these capabilities was real complicated during the development but the complexity is hidden behind the light sources and BSDFs implementations, so the code of the algorithm is relatively clean.

The implementation of the algorithm has many parameters that control its behavior. Next we provide a list of these parameters together with their description.

- **Initial radius** controls the impact of vertex merging. It can be specified in two ways. Either the user can provide it directly in the scene units or it can be derived from the radius of the scene bounding sphere. The later case is proposed in the original paper and the user just specify the percentage of the radius. We find this approach less intuitive and so we have implemented the first method as well.
- The user can specify the **radius reduction parameter** which is used to derive the radius for a particular algorithm iteration. The radius reduction scheme used in our implementation is described by the equation 3.8.
- **Russian roulette offset** can be specified to decrease the variance for short paths with potentially high measurement contribution.
- Beside the Russian roulette, the user can also control the **maximal depth** of a path in the number of its vertices. Value of zero lets the Russian roulette to bound the depth. This choice might be dangerous if non-physical BSDFs are present in the scene, since our implementation of Russian roulette depends on the albedo of a particular BSDF.
- **Maximal number of iterations** can be specified. This might be useful when performing a benchmark.
- The algorithm can be told by the user to run in the **bidirectional path tracing mode**. We have achieved this the same way as it is suggested in the technical report[Geo12]. This is useful if we want to compare the VCM with BPT or if we just know that VCM is not required for a particular scene and lighting conditions.
- **Merging at the second eye vertex** can be turned off by the user. Merging on this vertex corresponds to the photon mapping without the final gathering which has a slow convergence if the merging radius is large. We will compare results with merging at second camera vertex on or off in the next chapter.

The vertex connection and merging implementation within the GIRT library is represented by the class **VCM**, which is as direct subclass of **AbstractRenderer**. The implementation is capable of running the whole algorithm in parallel by the specified amount of threads. Now we continue by presenting its implementation of the **startRendering()** virtual method in the form of a pseudocode and literate programming.

What follows is the skeleton of the algorithm. At first, rendering process is initialized and then the VCM iteration loop begins, which consist in preparation, generation of the light paths, generating photon map (point search acceleration structure), and rendering. Note that if the algorithm runs in the BPT mode there is no need for the photon map, which is used exclusively for the vertex merging. Generating lights path in advance then unnecessarily wastes memory space, so in the case of the BPT mode turned on we generate light path right when it is required for an eye path sample.

```
⟨GIRT VCM Algorithm⟩ ≡
  ⟨Prepare The Rendering Process⟩
  while not cancel
  {
```

```
  ⟨Prepare VCM Iteration⟩
  if not BPT mode
  {
    ⟨Generate Light Paths and Photon Map⟩
    ⟨Build Range Query Data Structure⟩
  }
  ⟨Render⟩
}
```

Preparing the rendering process consist in clearing the framebuffer, computing the initial radius, clearing all state variables to some appropriate initial value, and reserving memory space. If the deducing of the initial radius from the scene bounding sphere radius is turned off no computation is done and the user specified radius is used directly. By reserving memory space we mean preallocating enough space for light paths and the photon map. Recall that the number of light paths traced is equal to the number of pixels, since the same paths are used for building the photon map and bidirectional path tracing estimate. If the resolution is high and the scene is a closed environment photon map can finally contain millions of photons. Not preallocating memory can result in a substantial performance degradation.

```
⟨Prepare The Rendering Process⟩ ≡
  ⟨Compute The Initial Radius⟩
  ⟨Reset State Variables⟩
  ⟨Clear the Framebuffer⟩
  ⟨Reserve Memory Space⟩
```

The preparation of a single iteration is mainly about increasing the iteration counter, then the merging radius for the upcoming iteration is evaluated using 3.8. At last some state variable and storage for light paths and photon map is cleared. Most of the state variables is helper variables that save computation later. For example there is the $\eta_{VCM} = \frac{nVM}{nVC}\pi r^2$ parameter that is used when computing every multiple importance sampling weight (VCM technical report eq. 20), where $r$ is the current merging radius, $n_{VM}$ is the number of vertex merging samples, and $n_{VM}$ is the number of vertex connection samples. Precomputing this parameter does not make the VCM interactive but is worthwhile.

```
⟨Prepare VCM Iteration⟩ ≡
  ⟨Apply Radius Reduction Scheme⟩
  ⟨Update State Variables⟩
  ⟨Clear Data⟩
```

Light paths generation is performed in parallel by the user specified number of threads. By default this number is equal to the number of available hardware threads and their priority is set below normal to keep the system responsive (especially Windows 7). The parallelization is realized using the helper class **AsynchronousTaskExecutor** which is a part of the GIRT library. An instance of this classes is more or less like a thread pool that is able to execute the given functor by required number of parallel threads up to some maximum. In the case of light paths generation the functor is a private method of the **VCM** class. This method generates light paths and photons in a loop batch by batch. After it generates a batch the function merges it with the global array within the **VCM** instance. The pseudocode describes the actual light paths generation procedure not the parallelization.

When storing a generated light path we do not consider the first vertex (on the light source). The first vertex is later generated new for every vertex of an eye path. This technique decreases the correlation between samples and can reduce the variance if specialized sampling techniques are employed. In the case of spherical luminaire we can use the point sampling technique described in the Siggraph course on Monte Carlo methods[Shi01a], which requires the knowledge of an eye path vertex.

```
⟨Generate Light Paths and Photon Map⟩ ≡
  ⟨Allocate Local Storage⟩
  while not cancel
  {
    if all paths generated
      break

    for i=1 to batchSize
    {
      ⟨Generate Light Path⟩
      ⟨Store Data To Local Structures⟩
    }

    ⟨Store Batch To Global Structures⟩
  }
```

Generation of a single light path begins with sampling a point on a light source using the probability density proportional to the emitted radiosity function. Then the first direction is sampled according to exitant radiance function at the chosen point. The particular light source is sampled according to its total emitted flux.

The light ray induced by the light sample is then traced through the scene and if it hits the scene surface, the first vertex (second actually) of the light path is initialized with all required parameters and stored. Path is then continued by the code that is common for generation of both path types (eye and light).

```
⟨Generate Light Path⟩ ≡
  ⟨Generate Light Sample⟩
  ⟨Trace the Ray Induced By the Light Sample⟩

  if light ray missed the scene
    return

  ⟨Initialize the Second Light Path Vertex⟩
  ⟨Continue Path Generation⟩
```

Continuation of the path generation is slightly more involved than everything we have seen so far. The procedure is based on a loop that is bounded by the maximum number of path vertices. At first the Russian roulette is possibly applied based on surface albedo for the current incident direction. If the path is not terminated by the Russian roulette a new direction is sampled using the BSDF at the current path vertex. The ray induced by the BSDF sample is traced through the scene and if it does not miss, a new path vertex is initialized with all its parameters. We should expand the fragment ⟨*Initialize New Path Vertex and Store It*⟩ further since it is the most complicated part of the procedure.

```
⟨Continue Path Generation⟩ ≡
  while not cancel && not max depth reached
  {
    if russian roulette should be applied
      ⟨Compute the Continuation Probability⟩
      if ξ > continuation probability
        break

    ⟨Sample Next Direction Using the Current Point's BSDF ⟩
    ⟨Trace the Ray Induced By the BSDF Sample⟩

    if BSDF ray missed the scene
      break

    ⟨Initialize New Path Vertex and Store It⟩
  }
```

The initialization of a new path vertex is mostly about computing the parameters for the recursive weight computation scheme. The procedure is not unified in this phase and it contains branches that are conditioned on the parameters of the previous and the current vertex. The goal of the following pseudocode is to show all different cases. Unfortunately, there is not enough space to show the details. In the case the reader is more interested we refer her or him directly to the source code and to the VCM implementation technical report[Geo12].

If the new sampled direction was proposed by a specular BSDF we have to handle the reverse sampling probability density explicitly, since the **BSDF** method that computes the density returns 0 if it is specular. The value of 0 is wrong in this case because we know that for our pair of directions the Dirac delta distribution that backs the specular BSDF would evaluate to $\infty$. See VCM technical report for the definition of reverse sampling probability density.

In the case the previous vertex has only specular BSDF we have to compute the parameters for the recursive MIS weight computation scheme differently, according to the equations 53-55 in the technical report, in order to disregard corresponding connections and mergings. In the other case we compute the parameters according to equations 31-36 in the technical report.

```
⟨Initialize New Path Vertex and Store It⟩ ≡
  ⟨Precompute Commonly Used Expressions⟩

  if sampled direction is specular
    ⟨Handle Reverse Probability Explicitly⟩

  if previous vertex is specular
```

```
    ⟨Specular Previous Vertex Parameters⟩
  else
    ⟨Non Specular Previous Vertex Parameters⟩
```

Having generated the records for the photon map, we can proceed with initialization of the range search acceleration structure. Since our merging radius is constant for all searches within a single iteration (a least for now), we can use a specialized structure for this kind of operation, the hash grid. We set the size of the grid box approximately to the size of the scene axis aligned bounding box and we set the cell size equal to the merging radius. This way the range query has constant complexity in the sense of the number of visited cells (eight). Parallelization of the hash grid building process is slightly more involved than the parallelization of the light paths generation. We used the Intel Thread Building Blocks[Int] for this task. See the source code of **PointHashGrid** for details. We should also mention that to save memory space our hash grid instance does not store the photons it has been built from. It stores only the world space positions and indices to the original array it has been given.

We can now move to the main function of the VCM algorithm procedure that performs the actual pixel measurements. As the light path generation the rendering is performed in parallel by the same amount of threads. Each thread processes a batch of pixel samples in a loop. The batch size is chosen large enough to avoid synchronization overhead as well as small enough to avoid a single thread running ten times longer just because it has been assigned a *complex* part of the image.

If the renderer is in the BPT mode we have to generate the light path for the current pixel sample using known procedure, otherwise we just fetch the stored light path from the global array.

Every light path vertex is firstly connected to the camera projection center. This connection is different from other vertex connections, since it can contribute to an arbitrary pixel in the image (light tracing).

If the light path generation was terminated by missing the scene geometry before the maximum amount of vertices has been reached or before the Russian roulette terminated it, we have to account for the eventual contribution of a distant area light source (environment map). This is a special case of unidirectional sampling but since distant area light can not be hit by ray tracing procedure, it has to be treated as a special case explicitly.

In the case an eye path vertex belongs to an area light source we have to account for the contribution of the corresponding path as in unidirectional path tracing.

As has been already mentioned, we handle bidirectional techniques $t \geq 2, s = 1$ by generating a new point on the light source. Other bidirectional techniques, $t \geq 2, s \geq 2$, are handled by connecting relevant points of the eye path and the light path. Vertex merging is performed starting at the second eye path vertex.

There is not enough space to provide a detailed description of every fragment mentioned in ⟨*Render*⟩. We believe that the actual source provided together with this thesis is clear and divided into short meaningful methods. The most complicated part of these fragments is MIS weight evaluation which is described by equations 38-47 in the VCM technical report[Geo12].

```
⟨Render⟩ ≡
```

```
while not cancel
{
  ⟨Acquire Pixel Range⟩
  for each pixel in the current range
  {
    ⟨Generate Eye Path⟩

    if BPT mode
      ⟨Generate Light Path⟩
    else
      ⟨Fetch Stored Light Path For the Pixel⟩

    for each light path vertex
      ⟨Connect Light Vertex To the Camera⟩

    ⟨Handle Environment Light⟩

    for each eye path vertex
      ⟨Connect Eye Vertex To the Light⟩
      ⟨Handle Unidirectional Contribution⟩

      for each light path vertex
        ⟨Connect Eye Vertex To Light Vertex⟩

      ⟨Merge Vertices at Eye Vertex⟩
  }
}
```

# Chapter 5

# Results

In this chapter we present an empirical evaluation of our vertex connection and merging implementation presented in the previous chapter. We start by discussing how we have checked the implementation's correctness. Then we analyze the results for various scenes and compare these results to other algorithms. The last section is devoted to the optimization of VCM parameters in the sense of different requirements on the image quality.

## 5.1 Testing correctness

To verify the correctness we implemented the path tracing with the next event estimation, which is relatively simple algorithm in comparison to the vertex connection and merging, and provide it with carefully designed input for which it can give the correct result in an affordable amount of time, which means that all non zero measurement contribution paths have relatively high probability density even if sampled by the path tracer. Assuming that the foundations of the GIRT library (e.g. BSDFs, sampling etc.) are implemented correctly and that our implementation of path tracing is done right, we can partially check the correctness of a new algorithm by comparing the result produced by it to the result produced by our path tracer. This was the first step we actually took to check our VCM implementation. Table 5.1 shows the result of the test. No tone mapping was applied and we can see that presented images are indistinguishable by a human eye. In fact the *root mean square* (RMS) difference of pixels luminances is 0.006. From the bottom left image of the table 5.1 we can see that this difference is caused only by a random noise in the images. There is no substantial difference in the form of missing some lighting effects.

We realize that checking our implementation through the comparison with the path tracing is not sufficient, since the VCM algorithm is designed to deal with the lighting conditions for which the path tracing is improper, because it either converges slowly or it does not converge at all. We will see such cases in the next section. For this reason we tested our implementation on such setup for which it was designed (*SDS* light carrying paths). We have also collected the scenes which were presented within the original VCM paper [GKDS12] and rendered them using our implementation. Then we compared our results with the results presented in the original paper.

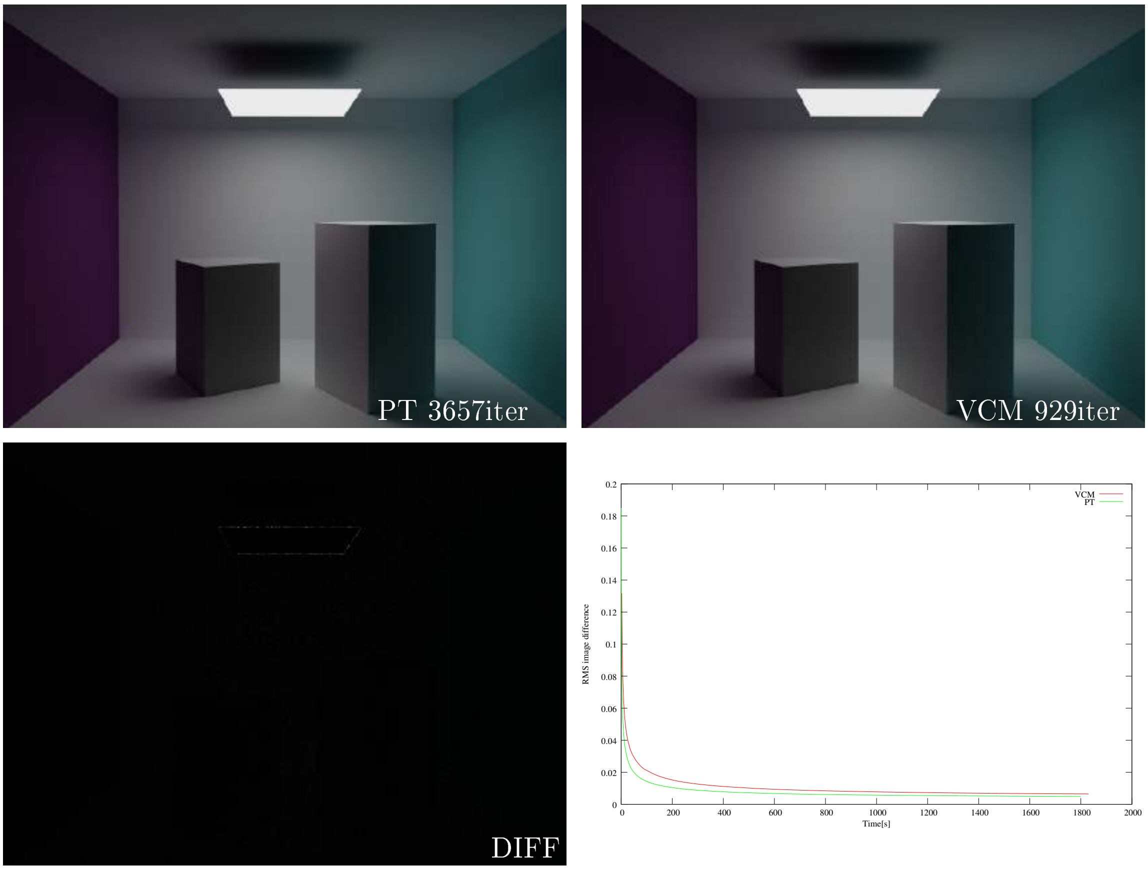


Table 5.1: The top left image shows the reference image produced by our implementation of path tracing with the next event estimation in 30 minutes. The top right image shows the same scene rendered by our implementation of vertex connection and merging in the same amount of time. The bottom left image visualizes the color difference between top images. The RMS difference is 0.006. The plot visualizes how the RMS difference from the reference evolved in time for both algorithms. The contour of the light source on the difference image is caused by the random sampling of pixel locations. See that the path tracer converges faster in this simple case (diffuse surfaces) than the VCM, since within 30 minutes it performed 3657 iterations while the VCM performed *only* 929 iterations.

Table 5.2: Tone mapped reference images of the scenes used for the evaluation of our VCM implementation. **Left:** *Mirror balls* scene by Toshiya Hachisuka **Middle:** *Living room* scene by Iliyan Georgiev **Right:** custom *Bathroom* scene

To properly test the correctness of a rendering algorithm one should provide it with an analytically computable input and then compare its result for the input with the analytically computed result. The problem with this approach is that the VCM algorithm was designed for cases for which the analytically evaluable example is hard to find. Not mentioning that our implementation might not pass due to the minor differences caused by 32 bit floating point arithmetic and otherwise negligible bias introduced by the recursive weight computation scheme [vA11]. Since our goal is photo realistic image we just need to be sure that our algorithm correctly simulates the global illumination effects it should and that relative pixel intensities are correct.

## 5.2 Rendering algorithms comparison

In this section we present an empirical evaluation of our vertex connection and merging implementation. We present the results for three selected scenes (table 5.2), though we have tested our implementation on more than just three. The appendix D contains noise free images of other scenes on which we have the VCM implementation. All scenes used for testing are located on the DVD-ROM provided together with this thesis.

All measurements were done on a single machine which specification is given below. Although our implementation uses the GPU only for displaying the resulting image we have included its specification.

- Intel Xeon E5-1620 CPU 3.60GHZ, 4 Cores, 8 Threads, 10MB Cache
- 16GB RAM, DDR3 1600MHz
- OS Windows 2008 R2 Server Edition 64bit
- NVIDIA GeForce GTX Titan (928MHz) 6GB DDR5 (6008MHz)

All presented images were rendered in the resolution of $1024 \times 768$ pixels. This implies that the VCM implementation traced 786 432 light paths in every iteration. Recall that in our implementation, the count of traced light paths is the same as the number of pixels in the image (see the previous chapter). The maximal depth of a single path was not specified, so the path generation was terminated at some point by the Russian roulette that was based on the albedo of the material at a surface point. This means that we had to make sure our scenes do not contain physically invalid materials. We tried to specify our materials with a great care but to make sure the algorithm will not end in an endless loop we ensured the physical validity directly in particular implementations of BSDF models. Radius reduction parameter $\alpha$ of the reduction scheme 3.8 for the vertex connection and merging and the bidirectional photon mapping (BPM) was always set to $\frac{2}{3}$ which is the optimal value proposed in the original paper [GKDS12]. Note that the VCM images presented in this section were created by performing the vertex merging till the third vertex of the eye path, disregarding it at the second one.

The table 5.5 shows images of all three scenes created by our VCM implementation in one minute and in one hour. For all images, the contribution of vertex merging techniques and the contribution of vertex connection techniques are shown separately. If a path, whose vertices are located at locations with relatively high photon density, is sampled by a vertex merging technique its contribution to the image will be relatively large due to high multiple importance sampling weight, especially when the merging radius is large. We can see that the vertex merging accounts for the reflected caustics. The directly visible caustics are handled by the light tracing, which is a vertex connection sampling technique. Recall that we do not perform vertex merging at the second eye path vertex. As the time grows the impact of the vertex merging decreases due to the merging radius reduction. One hour images show that most of the vertex merging image contribution comes from the reflected caustics only. Recall that reflected caustics are transported through the paths which have very low probability densities when sampled by the vertex connection techniques only. The vertex merging has a relatively large MIS weight for such path even when the merging radius is small.

The tables 5.6, 5.7, and 5.8 show a comparison of VCM to other global illumination algorithms, namely to the bidirectional photon mapping, to the bidirectional path tracing, and to the path tracing with the next event estimation. All mentioned algorithms were implemented within the GIRT library. The reason for the comparison of VCM with BPM and BPT should be clear. Recall that VCM is a combination of these algorithms. The comparison with path tracing is presented only to show the necessity of complex global illumination methods.

We can see that the path tracing is unable to account for certain lighting effects like reflected caustics or caustics caused by a small light source even if the next event estimation technique is used. The usage of multiple importance sampling for the direct lighting computation might improve the result but not substantially. This should serve as a good motivation for complex global illumination methods.

In overall, we can see that the bidirectional path tracing has problems in the areas with reflected caustics. Paths that transport the reflected caustics have a low probability density and if they happen to be sampled they only cause few bright pixels in the image, which appear as artifacts. In a reasonable amount of time (about one hour) the *CPU* implementation of BPT is unable to converge to the correct result, if reflected caustics paths are present.

The bidirectional photon mapping showed to produce visually more acceptable results than BPT in exchange for the bias. But for example if the illumination is caused by a distant light source photon density might get so low that the algorithm is unable to refine its result.

| Scene | I | IT | LGT | Merging | ALPL | AEPL | ACC | PC | MR |
|---|---|---|---|---|---|---|---|---|---|
| Mirror Balls | 410 | 8.5 | 1.5 | 2.1 | 7 | 6 | 41.5 | 4.07 | 3 |
| Living Room | 340 | 10.5 | 1.36 | 1.4 | 3.6 | 4.2 | 14.8 | 2.01 | 2 |
| Bathroom | 205 | 17.4 | 1.6 | 3.4 | 7.4 | 8.1 | 59.6 | 5 | 3.8 |

Table 5.3: This table shows some profiling information of the last VCM iteration that was performed when the images for the tables 5.6, 5.7, and 5.8 were created. The table 5.4 provides column descriptions.

| Column | Description |
|---|---|
| **I** | Number of VCM iterations done so far. |
| **IT** | Time of the last iteration. |
| **LGT** | Total time consumed by the light paths generation. |
| **Merging** | Average time spent with vertex merging by a single thread. |
| **ALPL** | Average light path length. |
| **AEPL** | Average eye path length. |
| **ACC** | Average number of vertex connections for a single pixel sample. |
| **PC** | Number of photons. |
| **MR** | Millions of rays per second. |

Table 5.4: This table provides column descriptions of the table 5.3. All times are in milliseconds and bound to the last iteration.

This is the case of the wall behind the mirror in the *Living room* scene. See that even after an hour of rendering the result is still *splotchy*.

The graphs presented within the tables 5.6, 5.7, and 5.8 show that bidirectional photon mapping converges to the reference image faster than other methods in the early phase of the rendering process but is always superseded by the vertex connection and merging after some time. Nevertheless, our tests showed that using VCM for a scene which can be efficiently handled by path tracing is not worth it. Even if the path tracer converges slower in the sense of the number of samples per pixel, it can evaluate a lot more samples than the VCM in the same amount of time (see the table 5.1).

The table 5.3 shows some profiling information of the last VCM iteration that was performed when the images for the comparison tables were created.

## 5.3 Optimizing VCM input parameters

As has been already mentioned, the original VCM paper [GKDS12] shown that the optimal value for the radius reduction parameter $\alpha$ is $\frac{2}{3}$. For this value the progressive photon mapping estimator has the optimal mean squared error (MSE) convergence rate of $O(1/N^{2/3})$. The MSE convergence rate of the vertex connection and merging then lies between the MSE convergence rate of the bidirectional path tracer $O(1/N)$ and the mentioned optimal rate of the progressive photon mapping.

A very important input parameter of the method that affects the visual appearance of the image in time is the initial merging radius. Choosing it too large results in the loss of detail due to the blurring by the density estimation kernel but the image looks smooth after relatively small amount of time. The blurred image can give the user an insight to the illumination of a scene which might be useful during its development. Choosing the radius too small might result in having the same problems as the bidirectional path tracing. The original paper proposed to derive the initial merging radius from the bounding sphere of the scene. When the scene is large and complex, we have found this approach a bit non intuitive, so in our implementation we allow the user to specify the initial radius directly in the scene units. The tables 5.10 and 5.11 show the results of the VCM method for various merging radii in time in the context of the *Living room* scene. To produce the table 5.10 we have turned the vertex merging at the second eye path vertex on and to produce the table 5.11 we have turned it off.

## 5.4 Increasing the number of vertex merging samples

As has been mentioned, the original paper [GKDS12] proposed that the number of vertex merging samples should be the same as the number of pixels in the image. We believe, this idea comes from the fact that as the resolution gets lower we are able to perform more vertex connection samples per pixel in the same amount of time, and as we have seen, the bidirectional path tracing has the better RMS convergence rate than bidirectional photon mapping. Nevertheless, we have modified our VCM implementation, so we can trace more photon paths than the number pixels (integer multiply of the number).

It has shown that sometimes it might be worth to increase the photon density as in the case of our test scenario, which is presented as the table 5.9. We used the *Living room* scene with the known environment setup from algorithm comparison and we have rendered this scene using three VCM configurations. In the first configuration we set the initial merging radius to 12.5mm and the number of photon paths to the number of pixels, in the second configuration we also set the merging radius to 12.5mm but set the number of photons paths fourth times the number of pixels, in the third configuration we set the merging radius to 25mm and set the number of photons path back to the number of pixels. Recall that all tests were done in the resolution of 1024x768 pixels. We let all of the mentioned VCM configurations to run for 60 minutes and all of them ended with more or less visually plausible result. We continue with the more elaborate comparison of the results.

By comparing first two configurations we wanted to see what happens if we increase the photon density for the same merging radius and it has shown that after 60 minutes the version that uses the increased number of photons paths has smaller RMS difference from the reference image than the version with the number of photon paths same as the number of pixels. We have also found the result of the higher density configuration visually better since the *splotches* caused by the vertex merging are not so accentuated.

By halving the merging radius and quadrupling the number of vertex merging samples (light paths), the MIS weight of a path proposed by a vertex merging technique should remain the same, which also means that overall contribution of vertex merging to the image should remain about the same. By looking at the results on table 5.9 for last two VCM configurations we can see that not only the RMS difference from reference is larger for the greater radius but also the result for greater radius contains noticeable low frequency noise.

This result showed that in some cases it might be worth to increase the impact of vertex merging techniques by increasing the number of its samples. For small radius the variance of bidirectional photon mapping estimator can be reduced substantially.

## 5.5 Computing image for more merging radii at once

Results of the VCM method for different merging radii brought the idea of computing images for more than one radius at once. We have modified our implementation so that we have extracted the merging radius from the recursive MIS weight computation scheme. This change is based on the VCM technical report [Geo12] where the weight computation scheme is modified to handle per vertex merging radii. We have measured the overhead of multiple radii on our test scenes. It showed up that the overhead is negligible in comparison to other operations. The overhead was always less then 0.3% of the total rendering time.

| **VCM** | **Vertex Merging** | **Vertex Connection** |
|---|---|---|

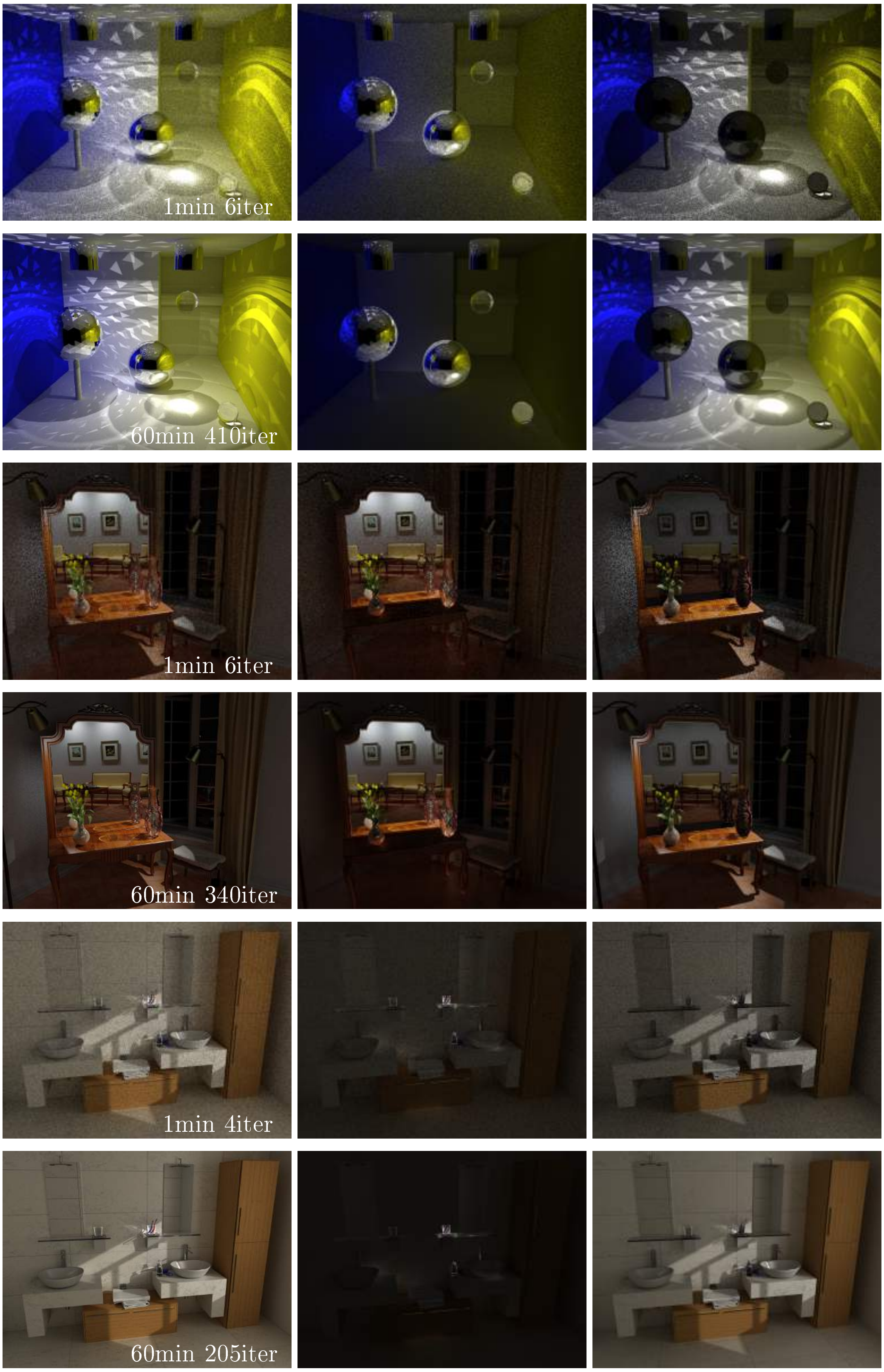


Table 5.5: The result of VCM rendering after 1 minute and after 1 hour of the three scenes used for the evaluation of our VCM implementation, Mirror balls, Living room, and Bathroom. First column shows the complete image while the middle column shows its vertex merging contribution and the right column shows its vertex connection contribution. The number of iterations accomplished by the algorithm so far is also provided.

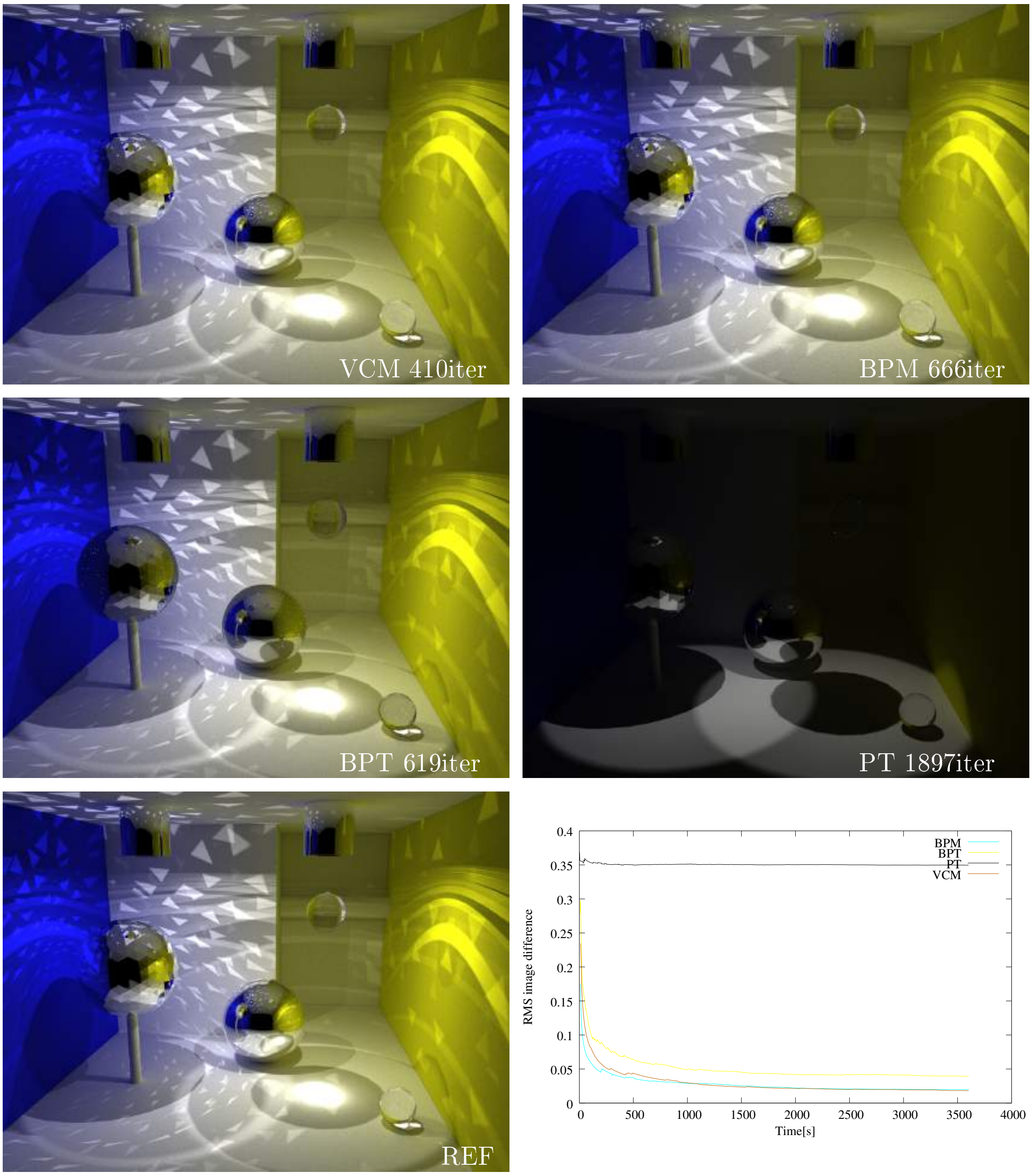


Table 5.6: The result of rendering the *Mirror balls* scene after 60 minutes by vertex connection and merging, bidirectional path tracing, bidirectional photon mapping, and path tracing with the next event estimation. Number of iterations (samples per pixel) is also provided. The bottom left image shows the reference image computed by the VCM algorithm in 5 hours. The plot in the bottom right corner shows the evolution of the RMS difference from the reference image for individual methods. The spikes in convergence of the VCM and the BPM were caused by the current merging radius and photons densities together at a particular time.

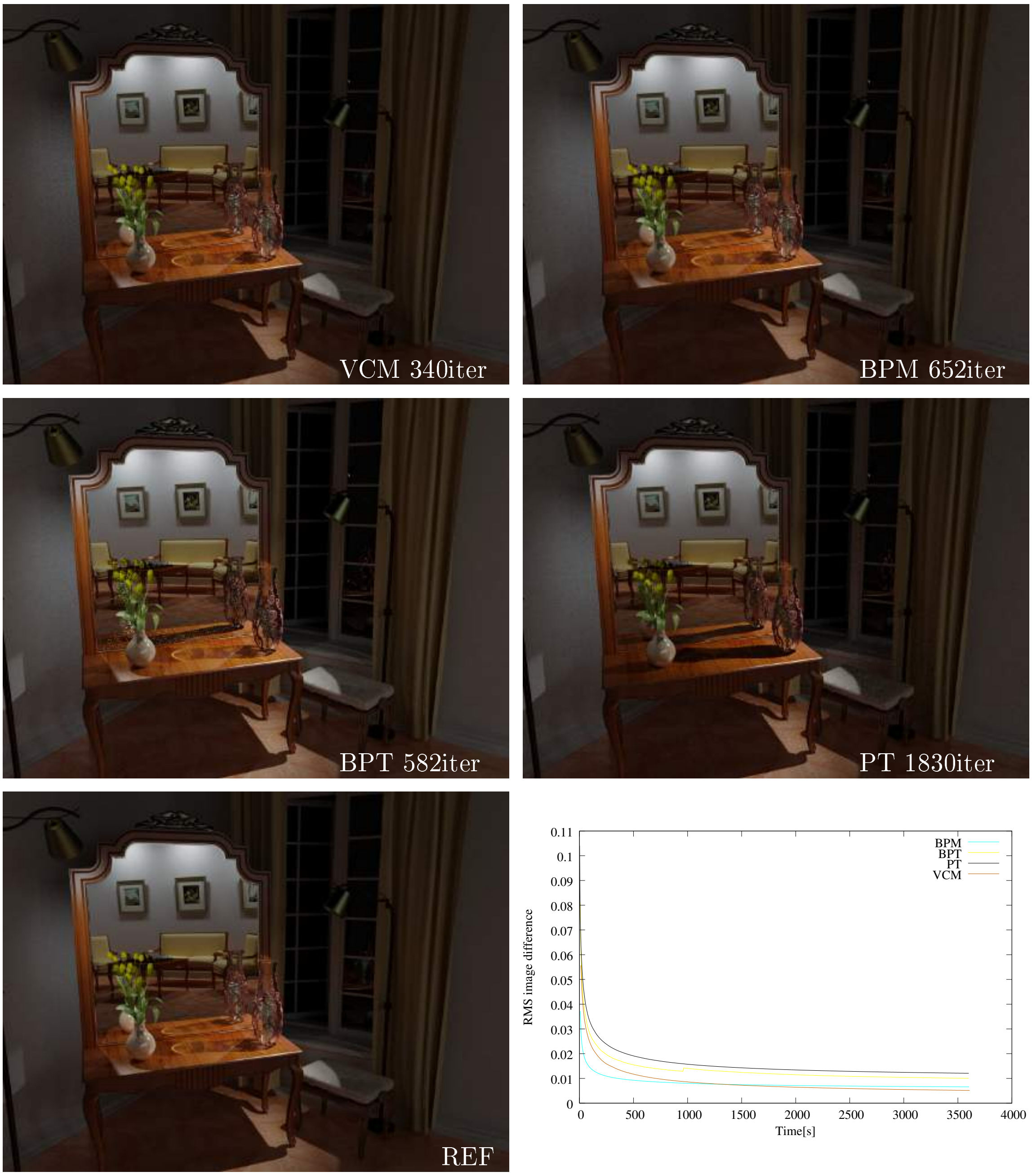


Table 5.7: The result of rendering the *Living room* scene after 60 minutes by vertex connection and merging, bidirectional path tracing, bidirectional photon mapping, and path tracing with the next event estimation. Number of iterations (samples per pixel) is also provided. The bottom left image shows the reference image computed by the VCM algorithm in 5 hours. The plot in the bottom right corner shows the evolution of the RMS difference from the reference image for individual methods.

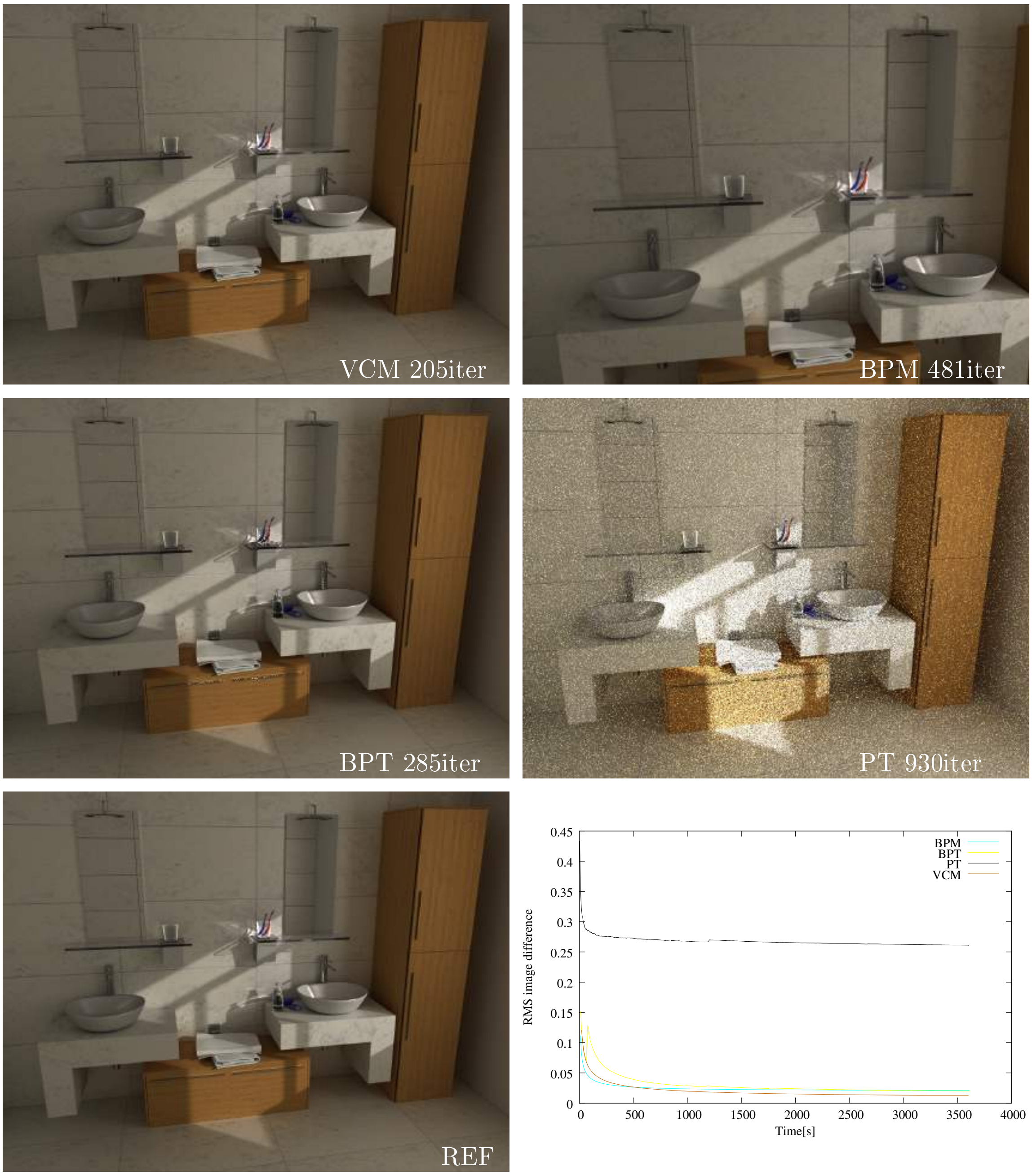


Table 5.8: The result of rendering the *Bathroom* scene after 60 minutes by vertex connection and merging, bidirectional path tracing, bidirectional photon mapping, and path tracing with the next event estimation. Number of iterations (samples per pixel) is also provided. The bottom left image shows the reference image computed by the VCM algorithm in 5 hours. The plot in the bottom right corner shows the evolution of the RMS difference from the reference image for individual methods.

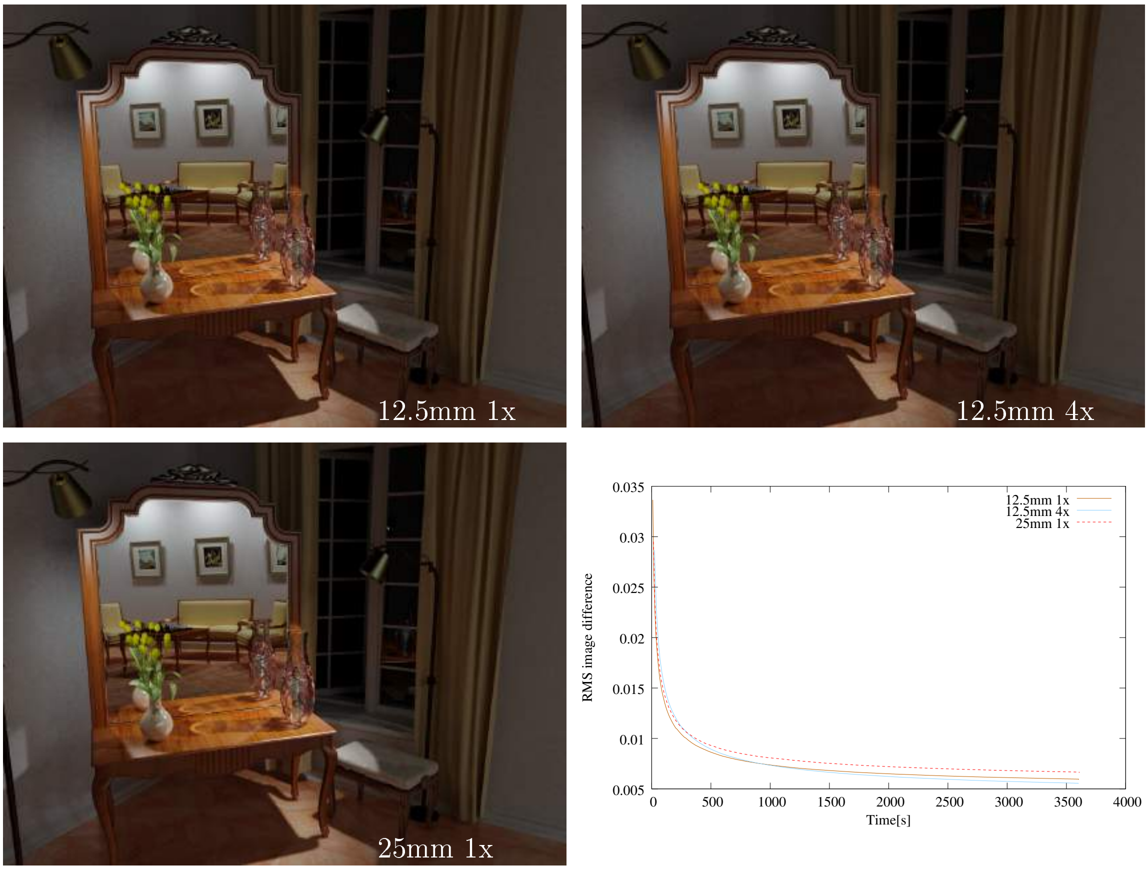


Table 5.9: Results presented in this table show that sometimes it might be worth to increase the impact of vertex merging by increasing the number of light paths, which most likely increases the amount of photon records. All images (1024x768) on this table were created by the VCM algorithm after 60 minutes. The top left image was created with the merging radius 12.5 mm and every iteration the number of pixels light paths were traced. The next image to the right was created with the same initial merging radius but the number of light paths were quadrupled. To generate the bottom left image we doubled the merging radius and changed the number of light paths back to the number of pixels. The plot shows the RMS difference from reference of all three images in time and we can see that increased number of photons led to a better result. See the section 5.4 for more detailed discussion.

| **r=5mm** | **r=12mm** | **r=25mm** |
|---|---|---|

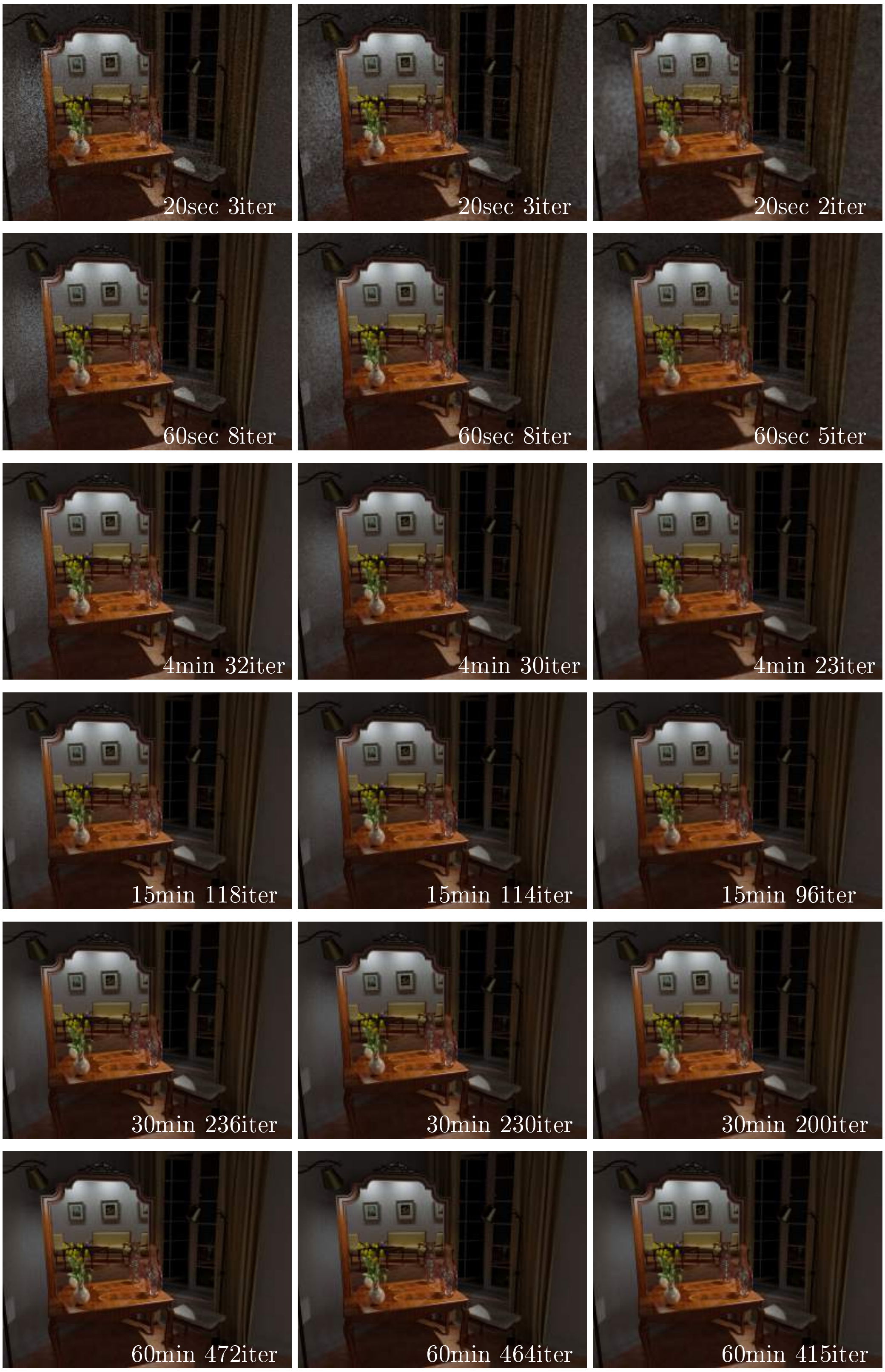


Table 5.10: The results of the vertex connection and merging for a various initial merging radii in time in the context of the *Living room* scene. The vertex merging at the second eye path vertex was turned on. The left column shows the result for the radius of 5mm, the middle column for 12mm, and the right column for 25mm.

**r=5mm** **r=12mm** **r=25mm**


Table 5.11: The results of the vertex connection and merging for a various initial merging radii in time in the context of the *Living room* scene. The vertex merging at the second eye path vertex was turned off. The left column shows the result for the radius of 5mm, the middle column for 12mm, and the right column for 25mm.

# Chapter 6

# Conclusion

Realistic image synthesis is a broad field of computer graphics and there are many different methods and approaches. The choice of a particular method depends on the requirements, whether we just want to compute a visually plausible image that contains some global illumination effects or whether we want the result to be physically accurate and indistinguishable from the photograph taken by a today's camera. This thesis presented theoretical background for any class of methods and discussed the algorithms suitable for photorealistic image synthesis.

We have successfully implemented and evaluated the state of the art method for global illumination, the vertex connection and merging. The implementation was realized within our global illumination rendering toolkit. It supports various BSDF models and many models of light sources, including delta distribution sources and HDR environment maps. It is capable of producing high quality images of relatively complex scenes. Some images produced by our implementation are provided in the Appendix D.

We checked the correctness of our vertex connection and merging implementation by comparing its result to the results of path tracing with the next event estimation with a positive result. Then the implementation was tested on three different scenes that exhibit difficult lighting conditions. It showed up that the VCM algorithm successfully converges to a visually plausible result as opposed to the result produced by the bidirectional photon mapping or bidirectional path tracing alone.

We have shown how the input parameters of the VCM algorithm affect the process of the image synthesis. The results might serve as motivation for further research, which can be based on them.

## 6.1 Future work

The success of a path sampling global illumination technique depends primarily on how efficient importance sampling it performs on the space of light carrying paths for a particular measurement. Even though we have seen vertex connection and merging create a visually plausible image in an affordable amount of time, it is not hard to imagine an input for which all presented methods fail. For example if we want to create an image of a single room within a building with many lit rooms, all presented bidirectional algorithms would fail by assigning a non zero probability to a large set of paths with zero measurement contribution. Metropolis light transport[VG97] tried to solve this problem but has its own known weaknesses, for example it might not find an important contribution to the image within an assigned time quantum. Using data from previous iterations to change the probability densities in the interesting portion of the path space might improve the convergence of the VCM method in a scenario like the one mentioned.

As we have seen, the result of the vertex connection and merging method highly varies with the initial radius for vertex merging. It is relatively cheap to compute the result for different radii simultaneously. The weight computation scheme has to be changed, so it does not contain the information about the current merging radius. This change is already described in the VCM technical report[Geo12]. One can then study how to combine the images of different radii in order to produce a visually plausible result.

Our implementation of the vertex connection and merging method provided the GIRT library with a powerful rendering tool, which might be a motivation for the improvement of the library itself in the sense of adding new ray casting acceleration data structures, BSDF models, light models etc. The actual implementation of the algorithm was written with the mentioned improvements of VCM in mind, so it can serve as a base for their realization.

# Bibliography


[Arv95] James Richard Arvo. *Analytic Methods for Simulated Light Transport*. PhD thesis, New Haven, CT, USA, 1995. AAI9619140.

[ATW] Anttweakbar. http://anttweakbar.sourceforge.net.

[BADC11] Francesco Banterle, Alessandro Artusi, Kurt Debattista, and Alan Chalmers. *Advanced High Dynamic Range Imaging*. A K Peters, Ltd., 1st edition, 2011.

[Bar95] Robert G. Bartle. *The Elements of Integration and Lebesgue Measure*. Wiley-Interscience, 1st edition, 1995.

[CK08] Marek Capinski and E. Kopp. *Measure, Integral and Probability*. Springer, 2008.

[CPC84] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. *SIGGRAPH Comput. Graph.*, 18(3):137–145, January 1984.

[Geo12] Iliyan Georgiev. Implementing vertex connection and merging. Technical report, Saarland University, 2012.

[GKDS12] Iliyan Georgiev, Jaroslav Křivánek, Tomáš Davidovič, and Philipp Slusallek. Light transport simulation with vertex connection and merging. *ACM Trans. Graph.*, 31(6):192:1–192:10, November 2012.

[HJ09] Toshiya Hachisuka and Henrik Wann Jensen. Stochastic progressive photon mapping. *ACM Trans. Graph.*, 28(5):141:1–141:8, December 2009.

[HOJ08] Toshiya Hachisuka, Shinji Ogaki, and Henrik Wann Jensen. Progressive photon mapping. *ACM Trans. Graph.*, 27(5):130:1–130:8, December 2008.

[HST13] Kaiming He, Jian Sun, and Xiaoou Tang. Guided image filtering. *IEEE Transcations on Pattern Analysis and Machine Intelligence*, 35(6), 2013.

[Int] Intel. Thread building blocks. https://www.threadingbuildingblocks.org/.

[Jen01] Henrik Wann Jensen. *Realistic Image Synthesis Using Photon Mapping*. A K Peters/CRC Press, 2nd edition, 2001.

[Jon] David Jones. Good practice in (pseudo) random number generation for bioinformatics applications. http://www0.cs.ucl.ac.uk/staff/d.jones/GoodPracticeRNG.pdf.

[Kaj86] James T. Kajiya. The rendering equation. *SIGGRAPH '86 Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, 20(4):143–150, 1986.

[LW93] Eric P. Lafortune and Yves D. Willems. Bi-directional path tracing. In *PROCEEDINGS OF THIRD INTERNATIONAL CONFERENCE ON COMPUTATIONAL GRAPHICS AND VISUALIZATION TECHNIQUES (COMPUGRAPHICS '93*, pages 145–153, 1993.

[LW94] Eric P. Lafortune and Yves D. Willems. Using the modified phong reflectance model for physically based rendering. Technical report, 1994.

[PH10] Matt Pharr and Greg Humphreys. *Physically Based Rendering: From Theory To Implementation.* Morgan Kaufmann, 2nd edition, 2010.

[Pho75] B. T. Phong. Illumination for computer generated pictures. *Communications of ACM*, 18(6):311–317, 1975.

[Ros09] Sheldon M. Ross. *A First Course in Probability.* Pearson, 8th edition, 2009.

[RWP+10] Erik Reinhard, Greg Ward, Sumanta Pattanaik, Paul Debevec, Wolfgang Heidrich, and Karol Myszkowski. *High Dynamic Range Imaging: Acquisition, Display, and Image-Based Lighting.* Morgan Kaufmann, 2nd edition, 2010.

[Sha03] Guarav Sharma. *Digital Color Imaging Handbook.* CRC Press LLC, 1st edition, 2003.

[Shi01a] Peter Shirley. Direct lighting via monte carlo integration. *State of the Art in Monte Carlo Ray Tracing for Realistic Image Synthesis*, pages 29–38, 2001.

[Shi01b] Peter Shirley. Fundamentals of monte carlo integration. *State of the Art in Monte Carlo Ray Tracing for Realistic Image Synthesis*, pages 15–30, 2001.

[TM98] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Proceedings of the Sixth International Conference on Computer Vision*, ICCV '98, pages 839–, Washington, DC, USA, 1998. IEEE Computer Society.

[vA11] Dietger van Antwerpen. Recursive mis computation for streaming bdpt on the gpu. Technical report, Delft University of Technology, 2011.

[Vea98] Eric Veach. *Robust Monte Carlo Methods for Light Transport Simulation.* PhD thesis, Stanford, CA, USA, 1998. AAI9837162.

[VG94] Eric Veach and Leonidas J. Guibas. Bidirectional estimators for light transport. In *Eurographics Rendering Workshop 1994 Proceedings*, pages 147–162, 1994.

[VG97] Eric Veach and Leonidas J. Guibas. Metropolis light transport. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, pages 65–76, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[Vor11] Jiří Vorba. Bidirectional photon mapping. *Proceedings of CESCG 2011*, 2011.

[WKB12] Bruce Walter, Pramook Khungurn, and Kavita Bala. Bidirectional lightcuts. *ACM Trans. Graph.*, 31(4):59:1–59:11, July 2012.

# Appendix A

# Materials definition file example

Listing A.1: Example of materials definition file

```
materials =
{
    #Array of materials, each materials is enclosed by { }
    {
        name = 'Wood',
        type = 'phong_modified',

        kd = #Diffuseness given by image texture
        {
            type      = 'image',
            wrap_mode = 'repeat',
            file      = 'textures/wood.jpg'
        },
        ks = #Specularity given by constant texture
        {
            type  = 'constant',
            value = (0.3, 0.3, 0.3)
        },
        shininess = #Specular exponent given by constant texture
        {
            type  = 'constant',
            value = 32
        }
    }
}
```

# Appendix B

# Lights definition file example

Listing B.1: Example of lights definition file

```
lights =
{
    {
        name     = 'CeilingLight',
        type     = 'area',
        Le       = (1.0, 1.0, 1.0),  #Constant radiance in all
            upper hemisphere directions
        scale    = 100.0,            #Radiance scale
        exponent = 0                 #Focuses the light around the
             normal
    },
    {
        name     = 'LampLight',
        type     = 'point',
        position = (4.3, 1.3, 2.0),
        color    = (1.0, 1.0, 0.7)
    },
    {
        name      = 'Sun',
        type      = 'dir',           #Directional light
        direction = (0.0, -1.0, 0.0),
        color     = (1.0, 1.0, 0.7),
        scale     = 10000
    }
}
```

# Appendix C

# Benchmark definition file example

Listing C.1: Example of a GIRay Benchmark configuration file

```
benchmark =
{
    resolution_w = 3840,
    resolution_h = 2160,
    scene_geometry_file = Bedroom.obj ,
    materials_file = materials.girt ,
    lights_file = lights.girt ,
    autosave_times = '20 1000 5000',
    exit_after_last_autosave = 0,
    render_system_config =
    {
        renderer =
        {
            type = VCM,
            initial_radius_scale = 0.04,
            radius_reduction = 0.666667,
            mode = vcm,
            initial_radius = 0.001,
            derive_initial_radius_from_scene = 1
        },
        camera =
        {
            vertical_FOV = 45,
            position = '(56.3231, 61.7524, -30.1371)',
            orientation = '(-0.0450603, 0.902451, 0.0973615,
               0.417492)'
        }
    }
}
```

# Appendix D

# Image gallery

This appendix presents tone mapped images of virtual scenes produced by our vertex connection and merging implementation. Vertex merging was turned off on the second eye path vertex.

Figure D.1: The virtual model of the hallway at Department of Computer Graphics at Czech Technical University in Prague create by Tomáš Kraus within his Bachelor's thesis. Rendering took 3 hours.


Figure D.2: The car model created by Alex Kuntz and modified by Iliyan Georgiev. The scene is lit by a HDR environment map. Render time was 1 hour.

Figure D.3: An image of the living room scene, which was modeled by Iliyan Georgiev. Render time was 2 hours.

Figure D.4: Bathroom scene lit by a special light source that imitates sunny daylight through a window. Rendering took 1 hour.

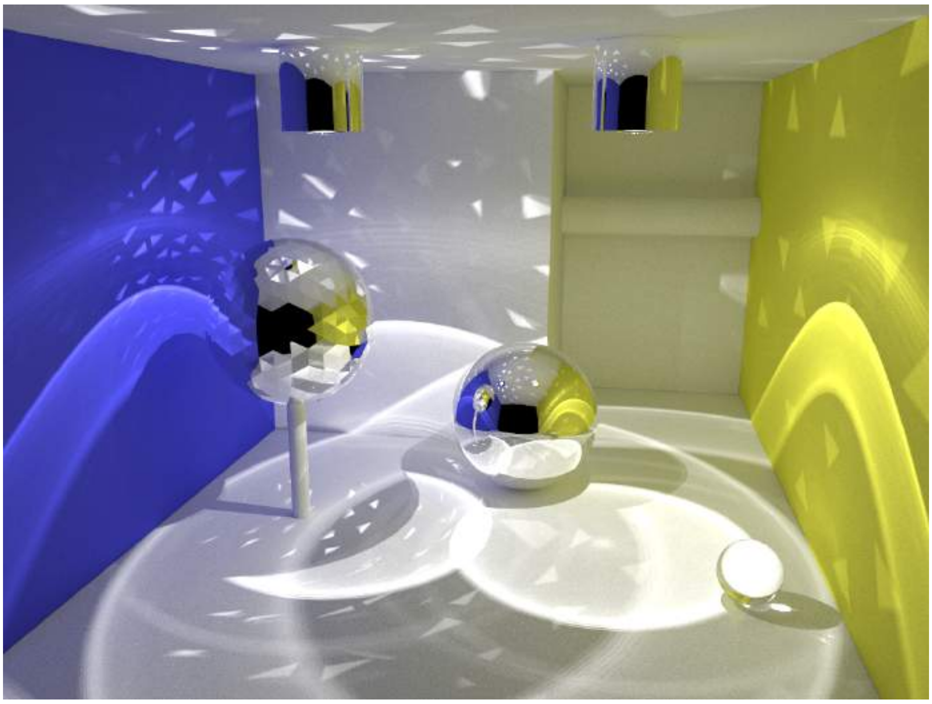

Figure D.5: Mirror balls scene by Toshiya Hachisuka rendered in 1 hour.

# Appendix E

# Installation and user manual

## E.1 Requirements

The following lists enumerates the important system requirements in order to run the binaries located in the `bin` directory.

- Microsoft Windows 7/8 64bit
- OpenGL, version 3.3 and higher

## E.2 GIRay application

GIRay is an interactive application based on the GIRT library presented in this thesis. The application contains real-time renderer which enables to set the view from which we want to create photorealistic image. It gives us the possibility to *interactively* change the parameters of the used rendering algorithm, see the intermediate result, and perform tone mapping and filtering on the final image. The optimized build of the application for OS Windows 7/8 64bit is provided in the `bin` directory on the DVD-ROM. The figure E.1 shows a screenshot of the application.

### Loading scene

GIRay loads the geometry of a scene from the *Wavefront OBJ* file. The materials have to be defined in the custom definition file `materials.girt` (see appendix A) or they can be defined by the *Wavefront MTL* file which has to be named same as the name of the scene geometry file. If both are present the former is used.

To load and render a scene in GIRay, perform the following steps:

1. Start GIRay by running `giray.exe` from the `bin` directory.
2. Press `Ctrl + O` and select the scene geometry file (`.obj`).

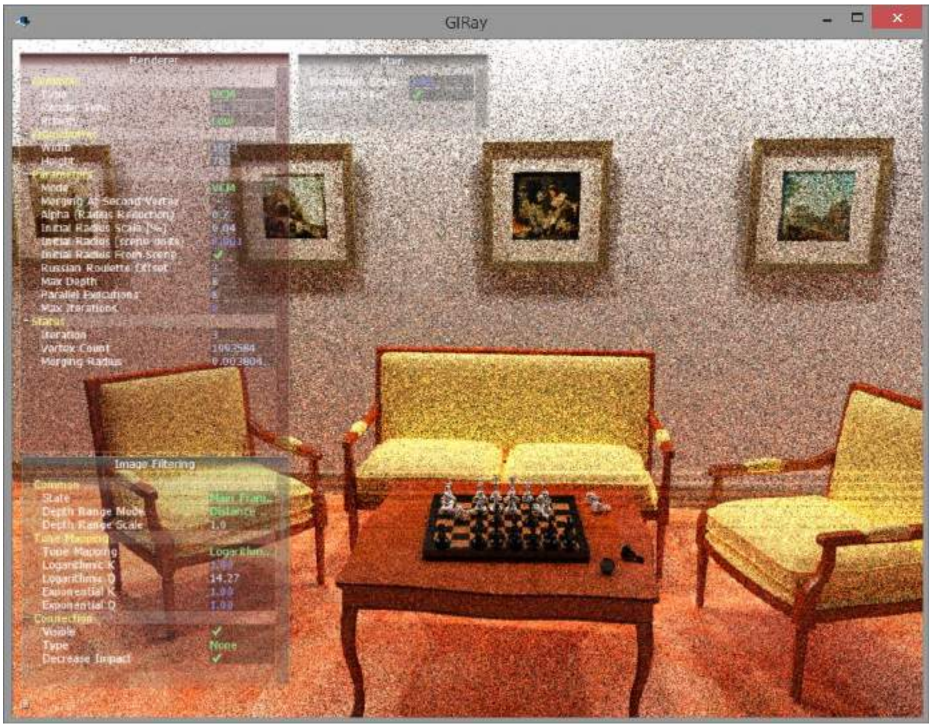


Figure E.1: A screenshot of the GIRay application.

### View manipulation

To change the view on the loaded scene you have to enter the view manipulation mode by pressing the `space` key. Then you can change the camera position by `W,S,A,D` keys and you can change the camera orientation using the mouse. The speed of the movement can be adjusted by the scroll wheel on the mouse. To exit the view manipulation mode you have to press `space` again.

If you want to save the current view on the scene use the choice *Save view* from context menu that is opened by pressing the right mouse button within the window of GIRay. If you save more than just one view you can switch between them by clicking on *Cycle view* in the same context menu.

### Settings

The application does not exploit the graphical user interface (GUI) components of the operating system to export the parameters of the GIRT library and it uses OpenGL based library AntTweakBar[ATW] instead. You can use this GUI to change the resolution of the resulting image, set the desired rendering algorithm (e.g. Path Tracer, VCM, etc.), change the parameters of the selected renderer (e.g. initial radius of vertex merging), select the image filter and adjust its parameters, or apply tone mapping.


Figure E.2: A screenshot of the GIRay application.

## E.3 GIRay Benchmark application

From the research point of view the GIRay Benchmark application is more important than GIRay. This application just loads a benchmark configuration file, which is discussed latter, and performs the rendering process using the GIRT library. It has a very simple GUI that just indicates the state of the benchmark. The figure E.2 shows a screenshot of the application.

### Benchmark configuration

The base of a GIRay Benchmark configuration file, whose example is given in the appendix C, should be created through GIRay. In GIRay, you set the view and the rendering algorithm and then you create the benchmark configuration by clicking on *Save benchmark* in the context menu.

Through the benchmark configuration you instruct the application when you want to save the state of the rendering process automatically (in seconds) and whether or not you want the application to exit after the last automatic save, which is good for performing benchmarks in batches. You can save the current rendering progress manually though the GUI of the GIRay Benchmark application.

The save of the rendering process consist in flushing the whole HDR framebuffer to the disk in the form of an *Industrial Light and Magic EXR* file. Every component of the framebuffer is saved to a separate *EXR* file. At the end of the benchmark a log file is created that contains the information about the passed rendering process. These information is collected by calling a specialized virtual method on the renderer used.

If you want the GIRay benchmark to compare the progress to a reference image, you have to provide the file named `reference.exr` beside the benchmark configuration file.

## E.4 Building from source code

### Requirements

The majority of requirements to build the GIRay applications is located in the `dependencies` directory. Nevertheless there are few dependencies that are not present in the directory since they are usually shared between many projects. These are:

- **CMake** The latest version of the CMake build system is required to create solution file for the Microsoft Visual Studio.

- **GLEW 1.7** This library is required by GIRay because it uses OpenGL 3.3 for image filtering.

- **wxWidgets 2.9.5** The wxWidgets library is used as a layer between OS and the GIRay applications.

### Build steps

1. Run the `build.but` script from the root directory which starts up the CMake build systems and points it to the `build` directory.
2. Select the generator. For example *Microsoft Visual Studio 10 Win64*, where *Win64* stands for the 64bit build.
3. Click on *Configure* button.
4. Eventually fix missing paths to dependencies (GLEW, wxWidgets).
5. Click on *Configure* button again.
6. Click on *Generate* button.
7. Open the generated VS solution in the `build` directory.
8. Select the *Build Configuration* (e.g. Debug, Release) and run the build.
9. Build the *INSTALL* project explicitly. That copies required *dll*s and OpenGL shaders to the `bin` directory.
10. You can now run the desired GIRay application.

# Appendix F

# DVD Content

```
hubrobin-thesis/        - root directory
|--bin/                 - destination for binaries
|--build/               - destination for the CMake build system files
|--cmake/               - custom scripts for the CMake build system
|--dependencies/        - 3rd party and custom libraries
|  |--AntTweakBar/      - AntTweakBar OpenGL GUI library
|  |--DevIL             - DevIL image manipulation library
|  |--PerfTools         - custom profiler
|  |--PObject           - library for loading PObject files
|  |--RapidXML          - XML parser required by PObject
|  |--TBB               - Intel Thread Building Blocks
|  |--VRUT              - compiled core of the VRUT
|--giray/               - source code root
|  |--app/              - source code of the GIRay application
|  |--benchmark_app/    - source code of the GIRay Benchmark application
|  |--girt/             - source code of the GIRT library
|--renders/             - HDR images rendered by our VCM implementation
|--results/              - results of our VCM implementation evalueation
|--scenes/              - scene files
```