Vertex Connection and Merging Based Techniques for Light Transport

(Techniki łączenia wierzchołków w symulacji oświetlenia)

Wojciech Szęszoł

Praca magisterska

Promotor: dr Andrzej Łukaszewski

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

16 maja 2018

Abstract

The aim of this work is research on the methods based on Vertex Merging and comparison of their relative performance against each other and against the classic path tracing algorithms. This work contains a description of the Vertex Connection and Merging and Unbiased Photon Gathering techniques from the theoretical and implementational point of view. The methods are the result of the recent research in the field of light transport simulation. The work also contains the results of the tests, that were carried on with the use of the aforementioned methods in various lighting conditions.

All of the methods described in this thesis were implemented and tested using the attached program, which is an integral part of the work. Every test image included in this thesis was generated using this software implementation.

Celem pracy jest zbadanie metod opartych na operacji łączenia wierzchołków (ang. Vertex Merging), porównanie ich wydajności względem siebie oraz względem klasycznych algorytmów śledzenia ścieżek. Niniejsza praca zawiera omówienie technik Vertex Connection and Merging (VCM) oraz Unbiased Photon Gathering (UPG) z teoretycznego i implementacyjnego punktu widzenia. Metody te są wynikiem ostatnich badań w dziedzinie metod symulacji oświetlenia. Praca zawiera także wyniki testów przeprowadzonych z użyciem wymienionych metod w różnych warunkach oświetleniowych.

Wszystkie metody opisane w pracy zostały zaimplementowane i przetestowane w załączonym programie, będącym integralną częścią pracy. Każdy z obrazów testowych zamieszczonych w pracy został wygenerowany z użyciem tego programu.

Contents

Chapter 1

Introduction

1.1 Problem Background

Visualization and rendering software has become a crucial tool in any modern engineering project. For some branches of the industry (automotive, architecture, entertainment, etc.) an important feature of the rendering system is precision and physical plausibility. This demand from the industry has been powering the research in the field of physically based rendering through the recent years. Lately developed light transport algorithms have an ability to handle arbitrary scenes, with complicated geometries, very general lighting conditions and complex surface properties.

The purpose of the rendering software is to simulate the transport of light through the mathematical representation of the scene. The description of the scene includes the positions and shapes of objects, appearance of their surfaces, positions of lights and cameras and many more. The theoretical foundation for any physically based rendering software is a rendering equation. The rendering equation is a mathematical formula that governs the way light propagates and interacts with the model of the scene. The light transport simulation is essentially equivalent to solving the rendering equation. Many different methods exist for solving the rendering equation. Since the rendering equation is fundamentally an integral equation, numerical integration methods are commonly used to solve it, in particular, a family of methods called Monte Carlo methods. The main topic of the research in this thesis are Monte Carlo light transport algorithms based on a novel concept of Vertex Merging ([GKDS12], [QSH+15a]).

1.2 Objectives

The aim of this thesis is to explore the Vertex Merging based algorithms, in particular, the recent developments in the field, presented in papers [GKDS12] and [QSH+15a]. The paper [GKDS12] introduced the Vertex Connection and Merging

algorithm. They found a novel way to combine Photon Mapping (PM) [Jen01] and Bidirectional Path Tracing (BPT) [VG94] under a single framework. The core idea behind VCM is Vertex Merging — a reinterpretation of the PM photon gathering step as a random event with an associated probability density function. The drawback of Vertex Connection and Merging (VCM) is that it is a biased algorithm. The work in [QSH+15a] builds on VCM and introduces an unbiased refinement of the initial algorithm, which is called Unbiased Photon Gathering (UPG).

1.3 Related Work

A comprehensive introduction into the field of physically based rendering and light transport simulation can be found in the excellent book by Matt Pharr and Greg Humpreys [PH10]. The work of Phillip Dutré [Dut03] contains a compact reference of mathematical formulas and equations commonly used in the context of light transport algorithms in computer graphics. The Pharr's book [PH10] and the initial chapters of the book by Henrik Wann Jensen [Jen01] give the background information about the physics of light and light scattering. Another very comprehensive work is Eric Veach's thesis [Vea97]. It contains theoretical fundamentals of the light transport simulation and the Monte Carlo methods.

The rendering equation and Path Tracing were introduced in 1986 paper [Kaj86] by James T. Kajiya. During early 90's, Eric P. Lafortune and Yves D. Willems developed, independently to Eric Veach and Leonidas J. Guibas, the Bidirectional Path Tracing (BPT) algorithm. Their initial research on BPT can be found in [LW93], [VG94], [LW94a] and [VG95]. The Veach's research on BPT led him to the development of the path integral framework, introduced in the mentioned PhD dissertation [Vea97] from December of 1997. In his thesis, Veach describes a provably efficient way to combine different BPT connections, that is a balance heuristic for multiple importance sampling (MIS). The tricky part of implementing BPT (and VCM) is an efficient implementation of the aforementioned balance heuristic weights. The 2011 paper by Dietger van Antwerpen [Ant11] contains the description of an efficient recursive computation scheme for mentioned weights. The van Antwerpen's recursive scheme is superior to the original scheme included in Veach's thesis when it comes to memory accesses. Additionally, the van Antwerpen's scheme is easier to integrate with the vertex merging.

Photon Mapping (PM) was an innovative algorithm for light transport simulation invented by Henrik W. Jansen. It is described in great detail in Jensen's book [Jen01] from 2001. Many improvements to the basic PM have been developed over the years following its invention. In the work [BfI03] from 2003, Philippe Bekaert and others proposed an improved photon gathering kernel and applied MIS to PM. In 2008 Toshiya Hachisuka and others developed a progressive, consistent version of PM in [HOJ08], and one year later, its refinement in [HJ09]. In 2012, Iliyan

Georgiev and others [GKDS12] invented a Monte Carlo reformulation of PT, allowing it to be combined with other path integral based techniques under the same multiple importance sampling framework. They proposed a Vertex Connection and Merging (VCM) algorithm, which is a combination of PM and BPT. The technical report [Geo12], which is a work supplemental to [GKDS12], contains a discussion about some of the issues with VCM implementation, including a description of how to apply van Antwerpen's MIS computation scheme to VCM. An alternative way to integrate PM with BPT under path the integral framework is described in the work, from the same year, by Toshiya Hachisuka and others [HPJ12].

The Vertex Connection and Merging is a biased algorithm, which is an unwanted property for some rendering applications. In recent years (2015), Hao Qin and others [QSH+15a] proposed an unbiased refinement of VCM. They invented an unbiased way, called Unbiased Photon Gathering (UPG), to estimate the probability density function of the vertex merging event.

1.4 Organization

The current, first chapter contains a brief description of what the thesis is about. The second chapter, after a short introduction about the Monte Carlo methods and the path integral, describes the basic BPT algorithm. After that the basic concepts of vertex merging techniques are introduced. Then, the VCM and UPG techniques are described from theoretical point of view. Sections 2.4.5 and 2.4.6 discuss the computation of the multiple importance sampling weights for VCM and UPG. The ending of the second chapter contains a description of the hash grid structure. The third chapter describes the software implementation of the mentioned algorithms. The initial sections describe the implementation itself in a closer detail, the latter sections talk about building and running the program. Two last sections are about the implementation correctness and its performance. The experimental results are presented in the fourth chapter. The last chapter contains a short summary and discussion about the possible extensions and future work. After the bibliography, there are two appendices (A and B), a snapshot from the command line interface of the program and a table with the errors for different gathering radii for the test scenes.

Chapter 2

Theory

2.1 Monte Carlo

2.1.1 Naive Monte Carlo

Multidimensional integrals which are present in the light transport simulation are problematic to solve using standard numerical integration methods due to their high dimensionality. Monte Carlo methods are the only known, reliable way to solve this kind of integrals in general settings [BfI03]. Consider a multidimensional definite integral on a subset $\Omega = [a_1, b_1] \times [a_2, b_2] \times \cdots \times [a_d, b_d]$ of $\mathbb{R}^d$:

$$I = \int_{a_1}^{b_1} \int_{a_2}^{b_2} \cdots \int_{a_d}^{b_d} f(x_1, x_2, \ldots, x_d) \, \mathrm{d}x_d \ldots \mathrm{d}x_2 \, \mathrm{d}x_1 = \int_{\Omega} f(\bar{x}) \, \mathrm{d}\bar{x}. \tag{2.1}$$

The basic Monte Carlo methods work by repeatedly evaluating integrated function f with random samples X_i drawn from uniform distribution over the integration interval Ω:

$$F_{MC} = \frac{v}{n} \sum_{i=1}^{n} f(X_i). \tag{2.2}$$

The essential property of the Monte Carlo methods is that the expected value of the estimator F_{MC} is equal to the value of the integral I (where $v = \prod_{i=1}^{d}(b_i - a_i)$ is the volume of the integration interval and n is the number of samples):

$$E\left[F_{MC}\right] = E\left[\frac{v}{n} \sum_{i=1}^{n} f(X_i)\right] = I. \tag{2.3}$$

What is more the variance of the estimator F_{MC} is equal to:

$$Var\left(F_{MC}\right) = E[(F_{MC} - I)^2] = \frac{1}{n}\left(v \int_{\Omega} f^2(\bar{x}) \, \mathrm{d}\bar{x} - I^2\right). \tag{2.4}$$

The above result has important implications. The first one is that using enough samples, any precision can be reached. The second one is that the RMS error convergence rate of Monte Carlo is proportional to $O\left(\sqrt{n}^{-1}\right)$, what in other words

means that to halve the error one needs to quadruple the number of samples. The third implication is that the convergence rate doesn't depend on the dimension of the integral (which is the major obstacle for using the conventional numerical integration techniques).

2.1.2 Importance Sampling

In practice the $O\left(\sqrt{n}^{-1}\right)$ convergence rate is very slow. The main focus of research on Monte Carlo methods is to improve it. There is nothing that can be done about $\sqrt{n}^{-1}$ factor in the RMS error. However, there are methods to improve the convergence rate by decreasing the other factors. One of such methods is the importance sampling. The importance sampling uses a customized probability density function p to concentrate the samples in the regions where the sampled function f has the largest values. A modified estimator is as follows:

$$F_{IS} = \frac{1}{n}\sum_{i=1}^{n}\frac{f(X_i)}{p(X_i)}, \tag{2.5}$$

and the expected value is, as before, equal to the integral:

$$E\left[F_{IS}\right] = E\left[\frac{1}{n}\sum_{i=1}^{n}\frac{f(X_i)}{p(X_i)}\right] = I. \tag{2.6}$$

The variance of the new estimator is equal to:

$$Var\left(F_{IS}\right) = E[(F_{IS} - I)^2] = \frac{1}{n}\left(\int_{\Omega}\frac{\left(f(x) - Ip(x)\right)^2}{p(x)}\,\mathrm{d}x\right). \tag{2.7}$$

The idea behind the importance sampling is to carefully choose the probability density function p. Notice that, if p was equal to $\hat{p}$, defined as:

$$\hat{p}(x) = cf(x) = \frac{f(x)}{\int_{\Omega} f(x)\,\mathrm{d}x}, \tag{2.8}$$

where c is the normalizing factor, ensuring that $\hat{p}$ integrates to 1, the value of the variance, and by extension the RMS error would be 0. Of course, it is impossible to use this theoretical probability density function $\hat{p}$, as it contains the integral of function f, which is the one that we want to approximate in the first place. However, the variance can be reduced by choosing such a p that its shape is similar to f.

2.1.3 Multiple Importance Sampling

Most of the time, it is hard to select the probability density function p that follows the shape of the integrated function f over the whole integration interval. Very often, we have the set of m sampling techniques and the probability density functions $p_1, p_2, ..., p_m$ associated with them. The particular function p_j may closely follow the shape of f in some regions of the integration interval but not in the others. Veach

introduced an estimator that combines all that different probability density functions in a way, that is provably close to the optimal [VG95] [Vea97]. The following equation presents a general form of the multiple importance sampling estimator:

$$F_{MIS} = \sum_{j=1}^{m} \frac{1}{n_j} \sum_{i=1}^{n_j} w_j(X_{i,j}) \frac{f(X_{i,j})}{p_j(X_{i,j})}. \tag{2.9}$$

The above estimator can be thought as linear interpolation between different importance sampling estimators. The factor p_j is a probability density function associated with the j-th technique. The random variable $X_{i,j}$ is independently sampled from distribution p_j. Weights $w_1, w_2, \ldots, w_m$ have to comply with following conditions, to guarantee that the estimator stays unbiased:

$$\begin{aligned} &\bullet \quad \sum_{j=1}^{m} w_j(x) = 1 \quad \text{when} \quad f(x) \neq 0 \\ &\bullet \quad w_j(x) = 0 \qquad \text{when} \quad p_j(x) = 0 \end{aligned}. \tag{2.10}$$

There are multiple weighting strategies to choose from. The power heuristic family of strategies is particularly good:

$$w_j(x) = \frac{n_j^{\beta} p_j^{\beta}(x)}{\sum_{k=1}^{m} n_k^{\beta} p_k^{\beta}(x)}. \tag{2.11}$$

Veach in his thesis [Vea97] proves that, in general circumstances, no other weighting strategy can be significantly better than the balance heuristic, which is the power heuristic with $\beta = 1$. He shows, as well, that the power heuristic with $\beta = 2$ is in practice a better choice. Notice, that with $\beta = 0$ the power heuristic, becomes a simple averaging scheme.

2.1.4 Russian Roulette

Given an estimator consisting of multiple terms, Russian Roulette [AK90] is a technique that allows to skip the computation of some terms of the estimator and get an unbiased result. Consider the following estimator with two terms:

$$F = F_H + F_T. \tag{2.12}$$

We want to conditionally skip the computation of the second term. Given the standard uniform random variable ξ and the probability p of executing the computation, we can replace the original estimator F with a Russian Roulette estimator F_{RR}:

$$F_{RR} = F_H + \begin{cases} \frac{F_T}{p} & \text{if } \xi < p \\ 0 & \text{otherwise} \end{cases}. \tag{2.13}$$

The expected value of modified estimator F_{RR} is the same as the expected value of initial estimator F. Clearly Russian Roulette increases the variance of the estimator its applied to. Nevertheless, it can improve efficiency, if it is used to skip the terms with low contribution and high computational cost. Additionally, Russian Roulette allows to compute estimators with infinite number of terms.

2.2 Rendering Equation

2.2.1 Rendering Equation

The rendering equation 2.14 describes the distribution of radiance (see Figure 2.1) in equilibrium state over all surfaces of the scene [Kaj86]. The classic formulation of the rendering equation assumes no participating media.

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_{\mathcal{S}^2} f_s(x, \omega_o, \omega_i) L_i(x, \omega_i) |\omega_i \cdot n| \, \mathrm{d}\omega_i \tag{2.14}$$

Given the surface position x in the scene and the normal vector n of surface at that position, $L_o(x, \omega_o)$ is the amount of the radiance outgoing in the direction ω_o from the point x. The term $L_e(x, \omega_o)$ is the radiance emitted by the surface itself in the outgoing direction. The integral part of the equation is called the scattering part and it represents the portion of light that is reflected by the surface. The factor $L_i(x, \omega_i)$ is the radiance incoming to the point x from the direction ω_i. The bidirectional scattering distribution function (BSDF) f_s is described in greater detail in Section 2.2.2.

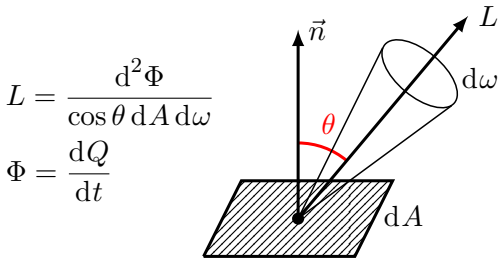

Figure 2.1: The radiance L is the radiant flux Φ by a given surface, per unit solid angle $\mathrm{d}\omega$, per unit projected area $\cos\theta \, \mathrm{d}A$. The radiant flux Φ is the flow of radiant energy per unit time $\mathrm{d}t$. The radiant energy Q can be viewed as the energy carried by the stream of photons.

In the absence of the participating media, there is a correspondence (see Equation 2.15) between the radiance incoming to the surface point x from the direction ω_i and outgoing from the surface point $x' = t(x, \omega_i)$ at the direction $-\omega_i$. The function $t(x, \omega_i)$ defines the first intersection with the scene of a ray, whose origin is x and which has the direction ω_i.

$$L_i(x, \omega_i) = L_o(t(x, \omega_i), -\omega_i) \tag{2.15}$$

Another version of the rendering equation, which is more suitable for the light transport simulation, uses integration over surface locations instead of the integration over the solid angle. Using the formula for the differential solid angle:

$$\mathrm{d}\omega = \frac{\omega' \cdot n'}{\|x' - x\|^2} \, \mathrm{d}A', \tag{2.16}$$

where n' is the surface normal at point x', the rendering equation becomes:

$$L_o(x,\omega_o) = L_e(x,\omega_o) + \int_{\mathcal{M}} f_s(x,\omega_o,\omega_i) L_i(x,\omega_i) V(x,x') \frac{|\omega_i \cdot n||\omega_i \cdot n'|}{\|x'-x\|^2} \,\mathrm{d}A'. \quad (2.17)$$

The factor $V(x,x')$ is a visibility function. The visibility function is equal to one when the points x and x' are mutually visible and zero otherwise. There is another form of the above equation, a so called three-point-form, it uses an explicit surface locations in place of the solid angles:

$$L(x_1 \to x_0) = L_e(x_1 \to x_0) + \int_A f_s(x \to x_1 \to x_0) L(x \to x_1) G(x \leftrightarrow x_1) \,\mathrm{d}A. \quad (2.18)$$

The factor $G(x \leftrightarrow x_1) = V(x \leftrightarrow x_1) \frac{|\omega_i \cdot n||\omega_i \cdot n_1|}{\|x_1 - x\|^2}$ is known as the geometry factor.

2.2.2 Bidirectional Scattering Distribution Function

The bidirectional scattering distribution function (BSDF) $f_s(x,\omega_o,\omega_i)$ models the interaction of the light with the surface. It is defined as the ratio between the amount of the reflected radiance and incoming irradiance:

$$f_s(x,\omega_o,\omega_i) = \frac{\mathrm{d}L_o(x,\omega_o)}{\mathrm{d}E(x,\omega_i)} = \frac{\mathrm{d}L_o(x,\omega_o)}{L_i(x,\omega_i)|\omega_i \cdot n| \,\mathrm{d}\omega_i}. \quad (2.19)$$

The BSDF is a generalization of the two other functions: the bidirectional reflectance distribution function (BRDF) and the bidirectional transmittance distribution function (BTDF). The BRDF and BTDF govern, respectively, how the light is reflected and transmitted through the surface.

Physically correct BSDF functions have to conserve energy, that is:

$$\int_{\mathcal{S}^2} f_s(x,\omega_o,\omega_i)|w_i \cdot n| \,\mathrm{d}\omega_i \leq 1. \quad (2.20)$$

Additionally, BRDFs obey the Helmholtz's law of reciprocity (Equation 2.21). In general BTDFs are not reciprocal. However, there is a generalization of this symmetry condition for BSDF, which have to be obeyed by BTDFs as well (Equation 2.22, [Vea97]).

$$f_r(x,\omega_o,\omega_i) = f_r(x,\omega_i,\omega_o) \quad (2.21)$$

$$\frac{f_s(x,\omega_o,\omega_i)}{\eta_o^2} = \frac{f_s(x,\omega_i,\omega_o)}{\eta_i^2} \quad (2.22)$$

2.2.3 Measurement Equation

The measurement equation 2.23 is an abstraction over an arbitrary measurement of radiometric quantity that can be done for the given scene [Vea97]. It is essentially an integration over all points x of the scene surfaces $\mathcal{M}$ and over the full sphere $\mathcal{S}^2$

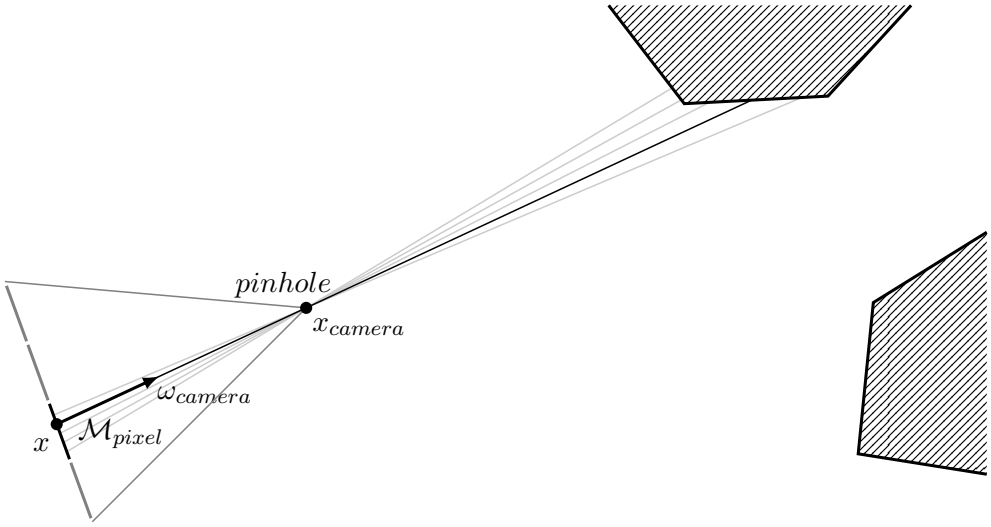

Figure 2.2: The sensitivity function for the pinhole camera selects rays with origin at the camera center and going through the pixel sensor.

of directions ω. The measurements are responses recorded by an abstract sensor, which is defined by its responsivity function. The responsivity function implicitly defines the shape of the sensor and its sensitivity.

$$I = \int_{\mathcal{M}\times\mathcal{S}^2} W_e(x,\omega) L_i(x,\omega) |\omega \cdot n| \,\mathrm{d}A \,\mathrm{d}\omega \tag{2.23}$$

Typically, we want to simulate the measurements done by some virtual camera. The camera consists of a flat sensor with grid of pixels and a lens system. The pixels of sensor corresponds to the pixels of the result image. Every pixel represents a separate measurement. In the context of the result image, the responsivity function is called a pixel sensitivity function. For simplicity, in the rest of the thesis, we will assume a pinhole lens with a flat rectangular sensor with square pixels. The pixel sensitivity function is unique for each pixel, as it is parametrized by its surface. The sensitivity function $W_e(x,\omega)$ of our sensor is equal to:

$$W_e(x,\omega) = f_p(x)\ \delta\left(\frac{x - x_{camera}}{\|x - x_{camera}\|} - \omega\right) |\omega_{camera} \cdot n|^{-1}, \tag{2.24}$$

where x_{camera} is the position of the virtual camera, $f_p(x)$ is 1 if x lies on the pixel surface and 0 otherwise (see Figure 2.2 for details). The inverse of the cosine of the angle between the w_{camera} and the normal n of the surface of the sensor is necessary to avoid the vignetting effect. Consider Figure 2.3. In order for the response of the upper sensor to be the same as the response of lower sensor the cosine factor needs to be canceled as the radiance from the surface on the right side is the same in both cases.

A measurement equation for the pinhole camera model is obtained by substituting the sensitivity function of the pinhole camera (Equation 2.24) into the general

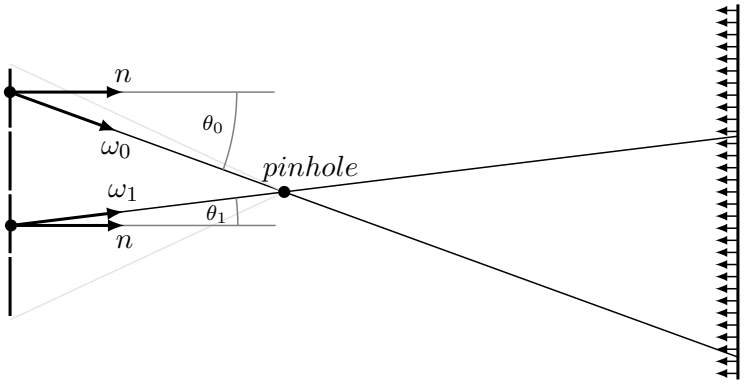

Figure 2.3: The avoid the effect of vignetting the radiance reaching the sensor must be scaled with cosine ($\cos\theta_i = \omega_i \cdot n$) of the incidence angle (with respect to the sensor plane) of the ray.

measurement equation (Equation 2.23):

$$I = \int_{\mathcal{M}_{pixel}} L_i(x, \omega_{camera}) \, \mathrm{d}A. \tag{2.25}$$

Now, the set $\mathcal{M}_{pixel}$ is the surface of the pixel sensor and $\omega_{camera} = \dfrac{x - x_{camera}}{\|x - x_{camera}\|}$.

2.2.4 Path Integral Framework

From the three-point form of the rendering equation it can be clearly seen that the equation is recursive, hence, it cannot be evaluated directly. Equation 2.26 presents its expansion.

$$\begin{aligned}
L(x_1 \to x_0) = {} & L_e(x_1 \to x_0) \\
& + \int_A L_e(x_2 \to x_1) f(x_2 \to x_1 \to x_0) G(x_2 \leftrightarrow x_1) \, \mathrm{d}A(x_2) \\
& + \int_A \int_A L_e(x_3 \to x_2) f(x_3 \to x_2 \to x_1) G(x_3 \leftrightarrow x_2) \\
& \qquad \cdot f(x_2 \to x_1 \to x_0) G(x_2 \leftrightarrow x_1) \, \mathrm{d}A(x_3) \, \mathrm{d}A(x_2) \\
& + \int_A \int_A \int_A L_e(x_4 \to x_3) f(x_4 \to x_3 \to x_2) G(x_4 \leftrightarrow x_3) \\
& \qquad \cdot f(x_3 \to x_2 \to x_1) G(x_3 \leftrightarrow x_2) \\
& \qquad \cdot f(x_2 \to x_1 \to x_0) G(x_2 \leftrightarrow x_1) \\
& \qquad \cdot \mathrm{d}A(x_4) \, \mathrm{d}A(x_3) \, \mathrm{d}A(x_2) \\
& + \dots
\end{aligned} \tag{2.26}$$

The terms in above equation correspond to the contributions of paths of different lengths. The equation is too verbose, so an additional notation is usually introduced. The product of BSDFs and geometry factors is called the path throughput and can

be written as:

$$T(\bar{x}_k) = \prod_{i=1}^{k-1} f(x_{i+1} \to x_i \to x_{i-1}) G(x_{i+1} \leftrightarrow x_i). \tag{2.27}$$

The variable $\bar{x}$ is a shortcut for a path of some length k: $\bar{x}_k = x_0 x_1 x_2 \dots x_{k-1}$. Now, the term for the path of length k from the recursive expansion of the rendering equation can be written as:

$$P(\bar{x}_k) = \underbrace{\int_A \int_A \cdots \int_A}_{\text{k-1}} L_e(x_{k-1} \to x_{k-2}) T(\bar{x}) \, \mathrm{d}A(x_2) \cdots \mathrm{d}A(x_n). \tag{2.28}$$

The expansion becomes an infinite sum of above terms:

$$L(x_1 \to x_0) = \sum_{k=2}^{\infty} P(\bar{x}_k). \tag{2.29}$$

Veach in his thesis [Vea97] introduced an elegant formulation, which combines the above derivation and the measurement equation. It is called the path integral framework. In the framework, the pixel measurement is represented as the integration over the space of light paths on the scene Ω:

$$I = \int_\Omega f(\bar{x}) \, \mathrm{d}\mu(\bar{x}). \tag{2.30}$$

The area-product measure space $\mathrm{d}\mu(\bar{x})$ is extended to the space of all light paths Ω. The function f is the measurement contribution function for a given pixel, and is defined in terms of the radiance emission, throughput and pixel sensitivity.

$$f(\bar{x}_k) = L_e(x_k \to x_{k-1}) \cdot T(\bar{x}) \cdot G(x_1 \leftrightarrow x_0) \cdot W_e(x_1 \to x_0) \tag{2.31}$$

2.2.5 Solving The Path Integral

The advantage of the path integral formulation is the Monte Carlo integration can be applied directly to it. The following equation shows an application of multiple importance scheme to estimate the value of the measurement I, where the measurement is expressed as the path integral:

$$I = \int_\Omega f(\bar{x}) \, \mathrm{d}\mu(\bar{x}) \approx \sum_{j=1}^{m} \frac{1}{n_j} \sum_{i=1}^{n_j} w_j(\bar{x}_{i,j}) \frac{f(\bar{x}_{i,j})}{p_j(\bar{x}_{i,j})}. \tag{2.32}$$

Equation 2.32 is the core of a wide range of modern light transport algorithms including the ones described in the further sections of this thesis. Every such an algorithm consists of m sampling techniques. The purpose of a technique is to efficiently sample a path x, that is to compute the measurement contribution f, the probability density p and the multiple importance sampling weight w associated with the given path. The contributions of multiple samples are then combined according to multiple importance sampling estimator, yielding an approximation of the final pixel value.

2.3 Vertex Connection Based Techniques

2.3.1 Bidirectional Path Tracing Overview

Bidirectional Path Tracing (BPT) is a light transport algorithm based on the path integral framework and multiple importance sampling. The initial version of BPT was introduced by Lafortune and Willems [LW93] [LW94a] and Veach and Guibas [VG94]. Then, the idea was refined with MIS by Veach and Guibas, in another paper [VG95] of theirs. Georgiev [GKDS12] coined the term Vertex Connection to refer to the central algorithmic step of BPT, which is connecting the end vertices of the light and camera sub-paths. The term was introduced to contrast with Vertex Merging. Vertex Merging is another approach to making a connection between the sub-paths, derived from Photon Mapping and its described in subsequent sections of this thesis. Hence, BPT is a Vertex Connection based technique.

The core idea behind BPT is to sample multiple paths, which connect a vertex on the light surface with a vertex on the camera lens (or the pinhole position for the pinhole camera model). The contributions of the samples are then accumulated in accordance with the multiple importance sampling estimator. The process is repeated until the desired accuracy is reached.

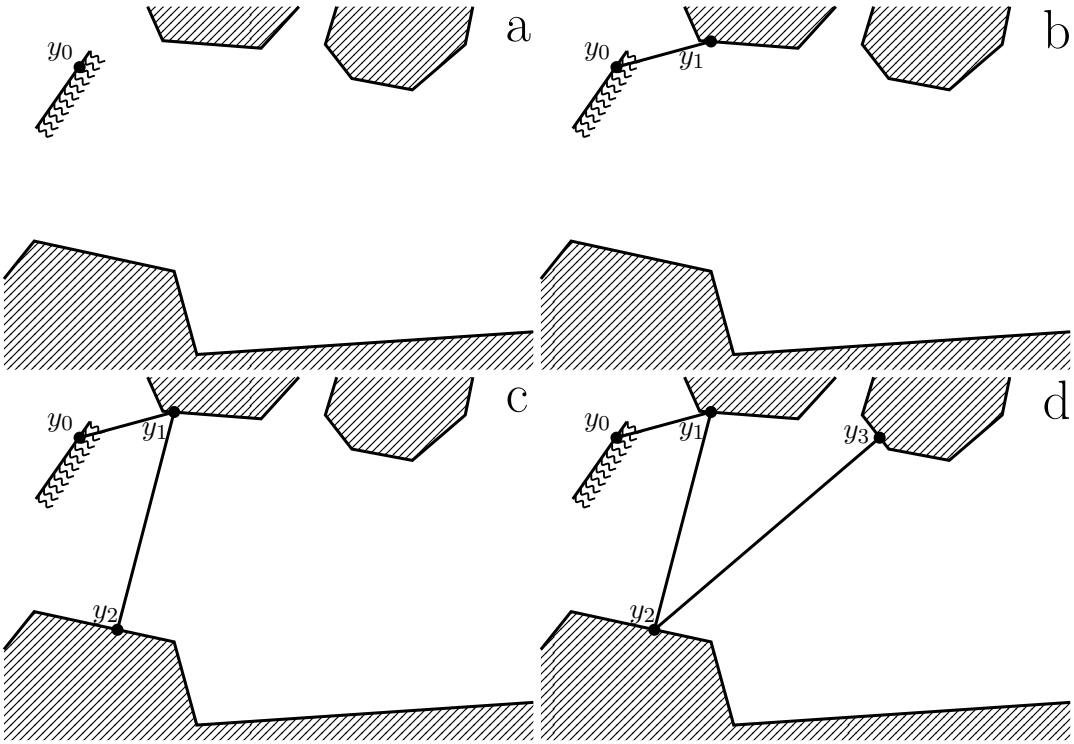

Figure 2.4: The first stage of BPT algorithm: tracing of the light sub-path.

The process starts from sampling of the light sub-path (see Figure 2.4). To sample the light sub-path a random position and direction at the light surface are chosen

(Figure 2.4a). The position will form the first light vertex y_0, and additionally, together with the direction they comprise a ray. In the next step, the first intersection of the ray with the scene is found (Figure 2.4b). This intersection becomes a position of a new sub-path vertex y_1. Then Russian Roulette is used to decide if the sub-path should be terminated. If no, the process is repeated to generate next vertices of the sub-path (Figure 2.4c and Figure 2.4d). Without Russian Roulette the process can continue for an arbitrarily long time, which is unacceptable due to the limited computational time. Aside from Russian Roulette the tracing of the sub-path may be terminated if the sub-path throughput (described later) reaches zero.

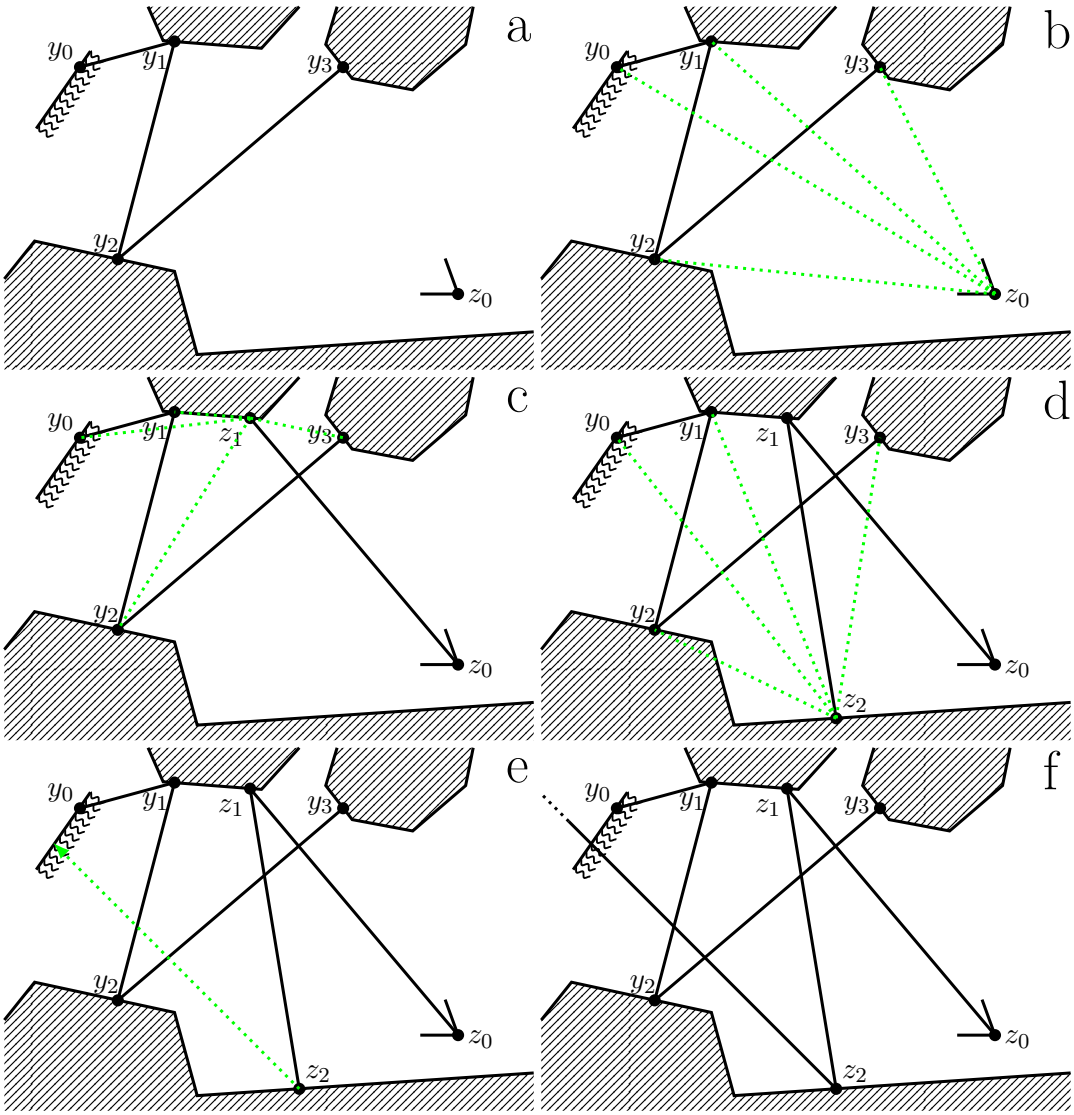

Figure 2.5: The second stage of BPT algorithm: tracing of the camera sub-path.

The next stage of BPT is tracing of the camera sub-path (see Figure 2.5). The overall scheme is the same as in the case of the light sub-path. However, before the camera sub-path is extended with another segment, its current ending needs to be connected (the connections are the dotted segments on Figure 2.5)

with each of the prefixes of the light sub-path. Additionally, before the intersection test with a non-emissive surfaces of the scene is done, the intersection with the light surfaces is examined (Figures 2.5e and 2.6). The reason for this, is to handle unidirectional camera paths. The unidirectional paths are important due to caustics. The paths with explicit BPT connection cannot handle delta distributions present in the specular reflections and transmissions. Furthermore, the contributions from the directly visible lights are handled by the unidirectional camera path with the length of one. The unidirectional path can be thought as if it was created by connecting the current camera sub-path prefix with the light sub-path prefix of zero length.

In general, the symmetric case, where the light path is traced unidirectionally from the light surface and hits the camera lens should be considered. However, with the assumption of a pinhole camera, this kind of paths can be safely ignored as the probability of hitting the sensor of the pinhole camera is equal to zero, so the contribution of such paths is zero as well.

The connection of the camera sub-path with any of the light sub-path prefixes produces a full light-camera path. Consider the path $\bar{x}_{s,t} = x_0 x_1 x_2 \dots x_{k-1}$. It is defined as the concatenation of the light sub-path $\bar{y}_s$ with the camera sub-path $\bar{z}_t$:

$$\bar{x}_{s,t} = x_0 x_1 x_2 \dots x_{k-1} = \bar{y}_s \bar{z}_t = y_0 y_1 y_2 \dots y_{s-1} z_{t-1} \dots z_2 z_1 z_0. \tag{2.33}$$

In order to add the contribution of the path to the final estimate, the measurement contribution function, the probability density function and the MIS weight of the path have to be computed (see Section 2.2.5). They can be efficiently evaluated by bookkeeping some extra data for each vertex of the light and camera sub-paths, while those sub-paths are traced.

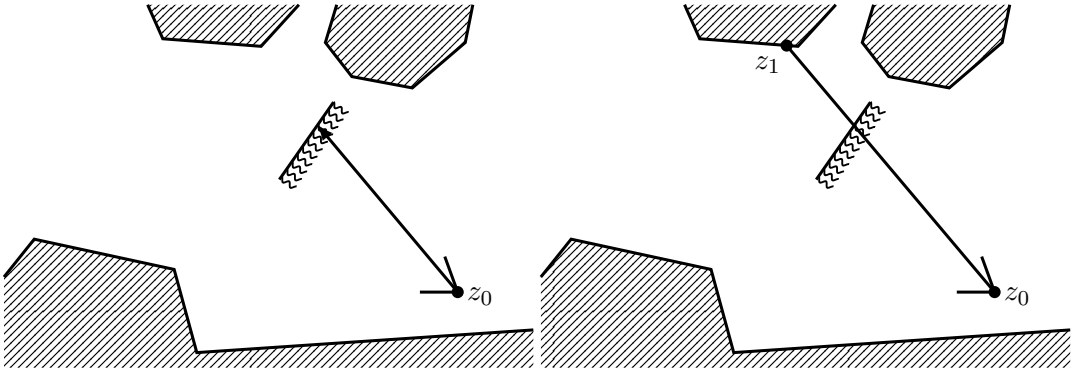

Figure 2.6: In order to include the contribution of the unidirectional paths, the intersection with the light surfaces is examined before the camera sub-path is extended with a new segment

It is important to notice that every path can be sampled in many different ways (see Figure 2.7). The different ways of sampling of the same path correspond to the multiple importance sampling techniques. There are $k + 1$ different techniques to sample the path of length k. Let's adapt the multiple importance estimator

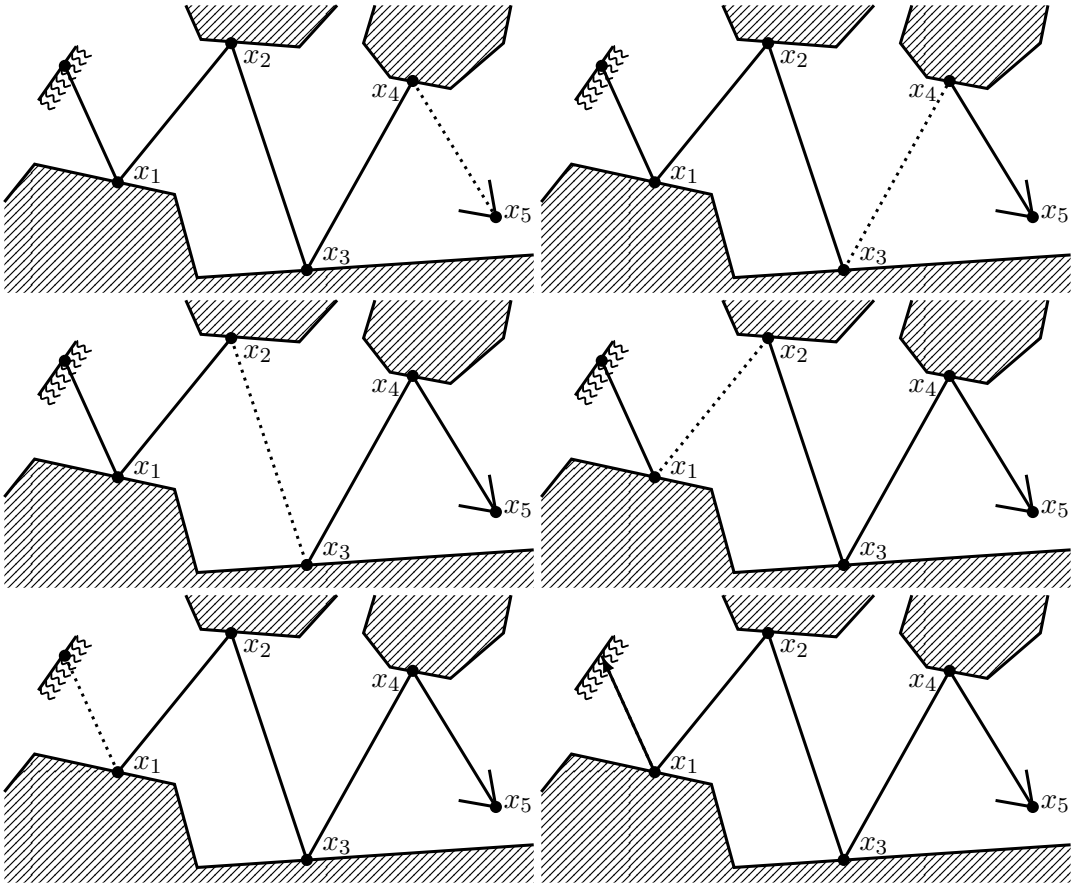

Figure 2.7: There are multiple ways to sample the same path. They differ in the place the connection is done. Some paths are sampled more efficiently by starting from camera and some by sampling from light. The path of length 5 has 6 sampling techniques.

(Equation 2.9) into the context of BPT:

$$F_{BPT} = \frac{1}{n}\sum_{i=1}^{n}\sum_{s\geq 0, t\geq 1} w(\bar{x}_{s,t})\frac{f(\bar{x}_{s,t})}{p(\bar{x}_{s,t})} = \frac{1}{n}\sum_{i=1}^{n}\sum_{s\geq 0, t\geq 1} w_{s,t}^{\bar{x}}\frac{f_{s,t}^{\bar{x}}}{p_{s,t}^{\bar{x}}}. \tag{2.34}$$

The first sum of the estimator corresponds to the multiple passes of the BPT algorithm. In every pass the sub-paths are traced and connected. The second sum accumulates the contributions of the full light-camera paths sampled by different techniques. The indexes s and t correspond to the lengths of the sub-paths. The index t starts from 1 because we ignore paths that directly hit the lens of the camera. In general, the light and camera sub-paths can be thought of as being infinite, however the sum can be evaluated in finite time as the paths are being terminated with Russian Roulette. In other words, due to Russian Roulette, the prefixes of sub-paths which include the Russian Roulette terminated vertex have throughput equal to zero.

2.3.2 Forward and Backward Factors

Before proceeding with explanation of certain factors in the BPT estimator, lets introduce some notation to simplify the formulas. The forward ($\vec{\cdot}$) and backward ($\overleftarrow{\cdot}$) partial geometry factors for the light ($\bar{y}$) sub-path are defined as:

$$\vec{g}_i^{\bar{y}} = \frac{\left|\vec{\omega}_i^{\bar{y}} \cdot n_i\right|}{\left\|y_i - y_{i-1}\right\|^2}, \quad \overleftarrow{g}_i^{\bar{y}} = \frac{\left|\overleftarrow{\omega}_i^{\bar{y}} \cdot n_i\right|}{\left\|y_{i+1} - y_i\right\|^2}. \tag{2.35}$$

The partial geometry factor for the first vertex of the path is explicitly defined as having a value of 1:

$$\vec{g}_0^{\bar{y}} = 1. \tag{2.36}$$

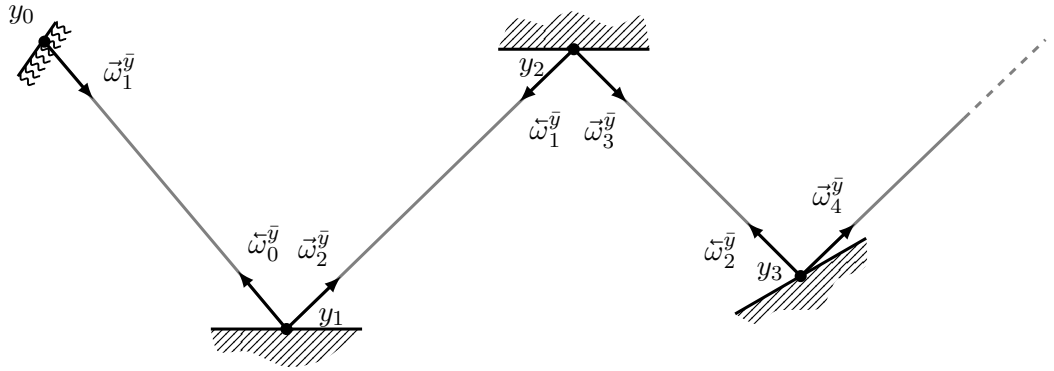

Figure 2.8: The forward and backward directions. The subscript denotes the vertex towards which the vector points. The arrow determines if the vector points along (forward, $\vec{\cdot}$) the direction of tracing the light sub-path or against (backward, $\overleftarrow{\cdot}$) the light sub-path.

To avoid confusion, in Equation 2.35 the normalized forward and backward directions ω were introduced. They are defined in the same fashion as forward and backward geometry factors:

$$\vec{\omega}_i^{\bar{y}} = \frac{y_i - y_{i-1}}{\left\|y_i - y_{i-1}\right\|^2}, \quad \overleftarrow{\omega}_i^{\bar{y}} = \frac{y_i - y_{i+1}}{\left\|y_{i+1} - y_i\right\|^2}. \tag{2.37}$$

Using the partial geometry factors (see Section 2.2.1 for a geometry factor) allows to define the forward and backward probability density functions for sampling the sub-path vertices. The probability density function for sampling a light sub-path vertex with respect to the area measure $\vec{p}_i^{\bar{y}}$ is defined as the product of the partial geometry factor $\vec{g}_i^{\bar{y}}$ and the probability distribution function with respect to the solid angle measure[1][2] $\vec{q}_i^{\bar{y}}$, as presented in Equation 2.38. The first vertices of the

[1]Notice that $\vec{q}_i^{\bar{y}}$ is probability of sampling the i–th vertex from the vertex y_{i-1}, or equivalently the sampling is done using the BSDF evaluated at the position of the vertex y_{i-1}. That fact is important from the implementational point of view, as to compute $\vec{q}_i^{\bar{y}}$ the data (position, normal, material etc.) associated with y_{i-1} (not with y_i) is needed. The same applies for the reverse $\overleftarrow{q}_i^{\bar{y}}$ factor.

[2]Usually $\vec{q}_i^{\bar{y}}$ is a probability density function following the shape of the corresponding BSDF.

corresponding sub-paths are special cases. The probability density function $\vec{p}_0^{\bar{z}}$ of sampling the vertex at the pinhole camera is equal to 1 (it is the only choice). For area lights the probability density function $\vec{p}_0^{\bar{y}}$ is essentially the inverse of the total area of all the area lights present in the scene (with the assumption that the lights are sampled uniformly). The forward and backward factors for the camera sub-path are defined in analogous way.

$$\vec{p}_i^{\bar{y}} = \vec{g}_i^{\bar{y}} \vec{q}_i^{\bar{y}},\ \overleftarrow{p}_i^{\bar{y}} = \overleftarrow{g}_i^{\bar{y}} \overleftarrow{q}_i^{\bar{y}},\ \text{for } i \neq 0. \tag{2.38}$$

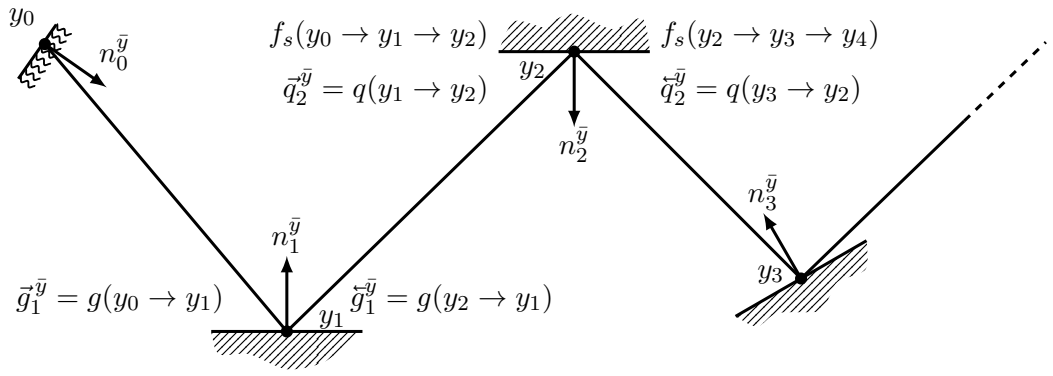

Figure 2.9: The concept of the forward and backward variables. Above the probability density functions for the vertex y_2, their corresponding BSDF functions are placed.

2.3.3 Measurement Contribution Function

The measurement contribution function[3] $f_{s,t}^{\bar{x}} = f(\bar{x}_{s,t})$ can be defined as:

$$\begin{aligned} f_{s,t}^{\bar{x}} &= L_e(x_{k-1} \to x_{k-2}) \cdot W_e(x_1 \to x_0) \\ &\cdot \prod_{i=1}^{k-1} (G(x_{i+1} \leftrightarrow x_i) f_s(x_{i+1} \to x_i \to x_{i-1})) \cdot G(x_1 \leftrightarrow x_0) \\ &= f_s^{\bar{y}} \cdot f_s(y_{s-2} \to y_{s-1} \to z_{t-1}) \cdot G(y_{s-1} \leftrightarrow z_{t-1}) \cdot f_s(y_{s-1} \to y_{t-1} \to z_{t-2}) \cdot f_t^{\bar{z}}. \end{aligned} \tag{2.39}$$

The symbols $f_s^{\bar{y}}$ and $f_t^{\bar{z}}$ stand for partial measurement contribution functions of the light and camera sub-paths $\bar{y}_s$ and $\bar{z}_t$ respectively. The function f_s uses the three point notation for the BSDF (see Equation 2.18). The sub-path measurement contribution functions are defined in a recursive way:

$$f_s^{\bar{y}} = \begin{cases} L_e^{(0)}(y_0) & \text{for } s = 0 \\ f_0^{\bar{y}} \cdot L_e^{(1)}(y_0 \to y_1) \cdot G(y_0 \leftrightarrow y_1) & \text{for } s = 1 \\ f_{s-1}^{\bar{y}} \cdot f_s(y_{s-2} \to y_{s-1} \to y_s) \cdot G(y_{s-1} \leftrightarrow y_s) & \text{otherwise} \end{cases}. \tag{2.40}$$

[3] See Section 2.2.4 and Equation 2.31.

The factors $L_e^{(0)}(y_0)$ and $L_e^{(1)}(y_0 \to y_1)$ require some explanation. The emitted radiance function $L_e(y_0 \to y_1) = L(y_0, \vec{\omega}_0)$[4] can be split into spatial $L_e^{(0)}(y_0 \to y_1)$ and directional $L_e^{(1)}(y_0 \to y_1)$ parts [Vea97]. They are defined as ($n_0^{\bar{y}}$ is the normal vector at the vertex y_0):

$$\begin{aligned} L_e^{(0)}(y_0) &= \int_{\mathcal{S}^2} L_e(y_0, \omega) \cdot \left| n_0^{\bar{y}} \cdot \omega \right| \mathrm{d}\omega \\ L_e^{(1)}(y_0 \to y_1) &= \frac{L_e(y_0, \overleftarrow{\omega}_0^{\bar{y}})}{L_e^{(0)}(y_0)}. \end{aligned} \tag{2.41}$$

The split is required to limit the number of special cases to consider in the implementation. Thanks to the split the directional part of the emitted radiance can be threated as a BSDF function:

$$L_e^{(1)}(y_0 \to y_1) = f_s(y_{-1} \to y_0 \to y_1). \tag{2.42}$$

As such, by the definition, the directional part obeys the Helmholtz's law of reciprocity:

$$\int_{\mathcal{S}^2} L_e^{(1)}(y_0 \to y_1) |w_i \cdot n| \, \mathrm{d}\omega = \int_{\mathcal{S}^2} \frac{L_e(y_0, \omega)}{L_e^{(0)}(y_0)} |w_i \cdot n| \, \mathrm{d}\omega = 1. \tag{2.43}$$

2.3.4 Probability Density Function

The probability of sampling of the full-path $\bar{x}_{s,t}$ is the product of the sampling probabilities for its light sub-path $\bar{y}_s$ and the camera sub-path $\bar{z}_t$, as the connection step is deterministic once the sub-paths are fixed. In turn, the sub-path probabilities are products of probabilities for sampling individual sub-path vertices with respect to the area measure:

$$p_{s,t}^{\bar{x}} = p_s^{\bar{y}} \cdot p_t^{\bar{z}} = \prod_{i=0}^{s-1} \vec{p}_i^{\bar{y}} \prod_{i=0}^{t-1} \vec{p}_i^{\bar{z}}. \tag{2.44}$$

The subsequent equation (2.45) presents the recursive formulation for the sub-path probabilities from the above formula. The recursive formulation is more suitable for the practical implementation.

$$p_s^{\bar{y}} = \begin{cases} \vec{p}_0^{\bar{y}} & \text{for } s = 0 \\ \vec{p}_s^{\bar{y}} \cdot p_{s-1}^{\bar{y}} & \text{otherwise} \end{cases}, \quad p_t^{\bar{z}} = \begin{cases} \vec{p}_0^{\bar{z}} & \text{for } t = 0 \\ \vec{p}_t^{\bar{z}} \cdot p_{t-1}^{\bar{z}} & \text{otherwise} \end{cases} \tag{2.45}$$

To save the memory and spare a few arithmetic operations the measurement contribution function and the probability density functions may be fused together into a single coefficient of the sub-path vertex (additionally, the partial geometry

[4] The $\vec{(\cdot)}$ notation means "forward" not "vector".

factors cancels out):

$$\alpha_s^{\bar{y}} = \frac{f_s^{\bar{y}}}{p_s^{\bar{y}}} = \begin{cases} \frac{L_e^{(0)}(y_0)}{\vec{p}_0^{\bar{y}}} & \text{for } s = 0 \\ \frac{f_0^{\bar{y}} \cdot L_e^{(1)}(y_0 \to y_1) \cdot V(y_0 \leftrightarrow y_1) \cdot \left| n_0^{\bar{y}} \cdot \vec{\omega}_0^{\bar{y}} \right|}{\vec{q}_1^{\bar{y}}} & \text{for } s = 1 \\ \frac{f_{s-1}^{\bar{y}} \cdot f_s(y_{s-2} \to y_{s-1} \to y_s) \cdot V(y_{s-1} \leftrightarrow y_s) \cdot \left| n_{s-1}^{\bar{y}} \cdot \vec{\omega}_{s-1}^{\bar{y}} \right|}{\vec{q}_s^{\bar{y}}} & \text{otherwise} \end{cases} . \tag{2.46}$$

2.3.5 MIS Weights

The last but not least piece to complete the BPT multiple importance sampling estimator are multiple importance sampling weights. The power heuristic (introduced in Section 2.1.3) is the weighting scheme usually used with BPT. The power heuristic weights for BPT are defined as:

$$w_{s,t}^{\bar{x}} = \frac{(p_{s,t}^{\bar{x}})^\beta}{\sum\limits_{s' \geq 0, t' \geq 1} (p_{s',t'}^{\bar{x}})^\beta} \overset{5}{=} \frac{(p_{s,t}^{\bar{x}})^\beta}{\sum\limits_{i=0}^{s+t-1} (p_{i,s+t-i}^{\bar{x}})^\beta} = \left(\sum_{i=0}^{s+t-1} \frac{(p_{i,s+t-i}^{\bar{x}})^\beta}{(p_{s,t}^{\bar{x}})^\beta} \right)^{-1} = \left(w_{s,t}^{-\bar{x}} \right)^{-1} . \tag{2.47}$$

For simplicity we will drop the β exponent in the following derivations[6]. Efficient evaluation of the power heuristic weights for BPT is quite complicated. The following scheme is based on the work of [Ant11] and the notes from [Geo12]. The main sum can be split into two sub-sums, which depend only on the values related to the light sub-path or camera sub-path respectively:

$$\begin{aligned} w_{s,t}^{-\bar{x}} &= \sum_{i=0}^{s+t-1} \frac{p_{i,s+t-i}^{\bar{x}}}{p_{s,t}^{\bar{x}}} = \sum_{i=0}^{s} \frac{p_{i,s+t-i}^{\bar{x}}}{p_{s,t}^{\bar{x}}} + \sum_{i=s+1}^{s+t-1} \frac{p_{i,s+t-i}^{\bar{x}}}{p_{s,t}^{\bar{x}}} = \sum_{i=0}^{s-1} \frac{p_{i,s+t-i}^{\bar{x}}}{p_{s,t}^{\bar{x}}} + 1 + \sum_{i=1}^{t-1} \frac{p_{s+t-i,i}^{\bar{x}}}{p_{s,t}^{\bar{x}}} \\ &= \sum_{i=0}^{s-1} \frac{\vec{p}_0^{\bar{y}} \cdots \vec{p}_{i-1}^{\bar{y}} \overleftarrow{p}_i^{\bar{y}} \cdots \overleftarrow{p}_{s-1}^{\bar{y}} \overleftarrow{p}_{t-1}^{\bar{z}} \cdots \overleftarrow{p}_0^{\bar{z}}}{\vec{p}_0^{\bar{y}} \cdots \vec{p}_{s-1}^{\bar{y}} \overleftarrow{p}_{t-1}^{\bar{z}} \cdots \overleftarrow{p}_0^{\bar{z}}} + 1 + \sum_{i=1}^{t-1} \frac{\vec{p}_0^{\bar{y}} \cdots \vec{p}_{s-1}^{\bar{y}} \overleftarrow{p}_{t-1}^{\bar{z}} \cdots \overleftarrow{p}_i^{\bar{z}} \vec{p}_{i-1}^{\bar{z}} \cdots \vec{p}_0^{\bar{z}}}{\vec{p}_0^{\bar{y}} \cdots \vec{p}_{s-1}^{\bar{y}} \overleftarrow{p}_{t-1}^{\bar{z}} \cdots \overleftarrow{p}_0^{\bar{z}}} \\ &= \sum_{i=0}^{s-1} \frac{\vec{p}_0^{\bar{y}} \cdots \vec{p}_{i-1}^{\bar{y}} \overleftarrow{p}_i^{\bar{y}} \cdots \overleftarrow{p}_{s-1}^{\bar{y}}}{\vec{p}_0^{\bar{y}} \cdots \vec{p}_{s-1}^{\bar{y}}} + 1 + \sum_{i=1}^{t-1} \frac{\overleftarrow{p}_{t-1}^{\bar{z}} \cdots \overleftarrow{p}_i^{\bar{z}} \vec{p}_{i-1}^{\bar{z}} \cdots \vec{p}_0^{\bar{z}}}{\vec{p}_{t-1}^{\bar{z}} \cdots \vec{p}_0^{\bar{z}}} \\ &= \sum_{i=0}^{s-1} \frac{\overleftarrow{p}_i^{\bar{y}} \cdots \overleftarrow{p}_{s-1}^{\bar{y}}}{\vec{p}_i^{\bar{y}} \cdots \vec{p}_{s-1}^{\bar{y}}} + 1 + \sum_{i=1}^{t-1} \frac{\overleftarrow{p}_{t-1}^{\bar{z}} \cdots \overleftarrow{p}_i^{\bar{z}}}{\vec{p}_{t-1}^{\bar{z}} \cdots \vec{p}_i^{\bar{z}}} \\ &= \sum_{i=0}^{s-1} \prod_{j=i}^{s-1} \frac{\overleftarrow{p}_j^{\bar{y}}}{\vec{p}_j^{\bar{y}}} + 1 + \sum_{i=1}^{t-1} \prod_{j=i}^{t-1} \frac{\overleftarrow{p}_i^{\bar{z}}}{\vec{p}_i^{\bar{z}}} . \end{aligned} \tag{2.48}$$

The partial sums can be rewritten in recursive manner. The following equation shows the expansion of the left hand side sum, which corresponds to the light sub-path.

[6] The power heuristic with $\beta = 1$ is called a balance heuristic.

The highlights unveil the recursion pattern:

$$\sum_{i=0}^{s-1}\prod_{j=i}^{s-1}\frac{\bar{p}_j^{\bar{y}}}{\vec{p}_j^{\bar{y}}}=\sum_{i=0}^{s-1}\prod_{j=i}^{s-1}\frac{\bar{g}_j^{\bar{y}}\bar{q}_j^{\bar{y}}}{\vec{g}_j^{\bar{y}}\vec{q}_j^{\bar{y}}}=\frac{\bar{g}_0^{\bar{y}}\bar{q}_0^{\bar{y}}}{\vec{g}_0^{\bar{y}}\vec{q}_0^{\bar{y}}}\cdot\frac{\bar{g}_1^{\bar{y}}\bar{q}_1^{\bar{y}}}{\vec{g}_1^{\bar{y}}\vec{q}_1^{\bar{y}}}\cdots\frac{\bar{g}_{s-3}^{\bar{y}}\bar{q}_{s-3}^{\bar{y}}}{\vec{g}_{s-3}^{\bar{y}}\vec{q}_{s-3}^{\bar{y}}}\frac{\bar{g}_{s-2}^{\bar{y}}\bar{q}_{s-2}^{\bar{y}}}{\vec{g}_{s-2}^{\bar{y}}\vec{q}_{s-2}^{\bar{y}}}\frac{\bar{g}_{s-1}^{\bar{y}}\bar{q}_{s-1}^{\bar{y}}}{\vec{g}_{s-1}^{\bar{y}}\vec{q}_{s-1}^{\bar{y}}}$$
$$+\frac{\bar{g}_1^{\bar{y}}\bar{q}_1^{\bar{y}}}{\vec{g}_1^{\bar{y}}\vec{q}_1^{\bar{y}}}\cdots\frac{\bar{g}_{s-3}^{\bar{y}}\bar{q}_{s-3}^{\bar{y}}}{\vec{g}_{s-3}^{\bar{y}}\vec{q}_{s-3}^{\bar{y}}}\frac{\bar{g}_{s-2}^{\bar{y}}\bar{q}_{s-2}^{\bar{y}}}{\vec{g}_{s-2}^{\bar{y}}\vec{q}_{s-2}^{\bar{y}}}\frac{\bar{g}_{s-1}^{\bar{y}}\bar{q}_{s-1}^{\bar{y}}}{\vec{g}_{s-1}^{\bar{y}}\vec{q}_{s-1}^{\bar{y}}}$$
$$+1\cdots\frac{\bar{g}_{s-3}^{\bar{y}}\bar{q}_{s-3}^{\bar{y}}}{\vec{g}_{s-3}^{\bar{y}}\vec{q}_{s-3}^{\bar{y}}}\frac{\bar{g}_{s-2}^{\bar{y}}\bar{q}_{s-2}^{\bar{y}}}{\vec{g}_{s-2}^{\bar{y}}\vec{q}_{s-2}^{\bar{y}}}\frac{\bar{g}_{s-1}^{\bar{y}}\bar{q}_{s-1}^{\bar{y}}}{\vec{g}_{s-1}^{\bar{y}}\vec{q}_{s-1}^{\bar{y}}}$$
$$\vdots$$
$$+\frac{\bar{g}_{s-3}^{\bar{y}}\bar{q}_{s-3}^{\bar{y}}}{\vec{g}_{s-3}^{\bar{y}}\vec{q}_{s-3}^{\bar{y}}}\frac{\bar{g}_{s-2}^{\bar{y}}\bar{q}_{s-2}^{\bar{y}}}{\vec{g}_{s-2}^{\bar{y}}\vec{q}_{s-2}^{\bar{y}}}\frac{\bar{g}_{s-1}^{\bar{y}}\bar{q}_{s-1}^{\bar{y}}}{\vec{g}_{s-1}^{\bar{y}}\vec{q}_{s-1}^{\bar{y}}}$$
$$+\frac{\bar{g}_{s-2}^{\bar{y}}\bar{q}_{s-2}^{\bar{y}}}{\vec{g}_{s-2}^{\bar{y}}\vec{q}_{s-2}^{\bar{y}}}\frac{\bar{g}_{s-1}^{\bar{y}}\bar{q}_{s-1}^{\bar{y}}}{\vec{g}_{s-1}^{\bar{y}}\vec{q}_{s-1}^{\bar{y}}}$$
$$+\frac{\bar{g}_{s-1}^{\bar{y}}\bar{q}_{s-1}^{\bar{y}}}{\vec{g}_{s-1}^{\bar{y}}\vec{q}_{s-1}^{\bar{y}}}. \tag{2.49}$$

Subsequent equation (2.50) presents the new formula for the weight, where A_{s-1} corresponds to the factors in the region with the red background in Equation 2.49 and a_{s-1} corresponds to the expression in the blue region.

$$w_{s,t}^{-\bar{x}}=(A_{s-1}\bar{q}_{s-2}^{\bar{y}}+a_{s-1})\bar{g}_{s-1}^{\bar{y}}\bar{q}_{s-1}^{\bar{y}}+1+(C_{t-1}\bar{q}_{t-2}^{\bar{z}}+c_{t-1})\bar{g}_{t-1}^{\bar{z}}\bar{q}_{t-1}^{\bar{z}}, \tag{2.50}$$

The sub-sum which corresponds to the camera sub-path can be expanded in the same way, however, a special attention needs to be put to the fact that the sub-path with $t=0$ is ignored. The regions in Equation 2.49 enclosed with the dashed lines correspond to the prior step of the recursion. The exact definitions for all of the variables go as follows:

$$\begin{aligned}
a_i &= \frac{1}{\vec{g}_i^{\bar{y}}\vec{q}_i^{\bar{y}}}\\
A_i &= \begin{cases} 0 & \text{for } i=0\\ (A_{i-1}\bar{q}_{i-2}^{\bar{y}}+a_{i-1})\bar{g}_{i-1}^{\bar{y}}a_i & \text{for } i>0 \end{cases}\\
c_i &= \begin{cases} 0 & \text{for } i=0\\ \dfrac{1}{\vec{g}_i^{\bar{z}}\vec{q}_i^{\bar{z}}} & \text{for } i>0 \end{cases}\\
C_i &= \begin{cases} 0 & \text{for } i=0\\ (C_{i-1}\bar{q}_{i-2}^{\bar{z}}+c_{i-1})\bar{g}_{i-1}^{\bar{z}}c_i & \text{for } i>0 \end{cases}.
\end{aligned} \tag{2.51}$$

Notice that both the base case and the general case C_i for 1 are equal to 0:

$$C_1=(C_0\bar{q}_{-1}^{\bar{z}}+c_0)\bar{g}_0^{\bar{z}}c_1=(0\cdot\bar{q}_{-1}^{\bar{z}}+0)\bar{g}_0^{\bar{z}}c_1=0. \tag{2.52}$$

2.3.6 Efficient Evaluation of the BPT Estimator

An observation can be made that the variables a_i and A_i (and c_i together with C_i as well, for the camera sub-path) depend only on the variables associated with i-th vertex of the light sub-path and additionally A_i depends on the variables A_{i-1} and a_{i-1} from the previous recurrence step[7]. It means that they can be efficiently (in constant time for each sub-path vertex) computed as the light sub-path is traced. Then the full weight can be evaluated, as well, in constant time using Equation 2.50. This fact is quite important, as naive evaluation of Equation 2.47 is quadratic and requires access to all vertices of the path at once.

This argumentation also applies to the probability density function and the measurement contribution function for the full light-camera path. Their recursive formulations were presented in the preceding sections. Hence, the computational cost of evaluating the BPT estimator for single connection of the sub-paths is constant. Since, every prefix of the light sub-path is connected to every prefix of the camera sub-path the computational cost is equal to the product of the number of the vertices of the light sub-path and of the camera sub-path. In practice it means the complexity is quadratic with respect to the total number of vertices on both sub-paths.

2.4 Vertex Merging Based Techniques

2.4.1 VCM and UPG Overview

Vertex Connection and Merging (VCM) is an algorithm introduced in [GKDS12]. It is a hybrid algorithm, which combines the ideas from Photon Mapping (PM, see [Jen01]) and Bidirectional Path Tracing (BPT, see Section 2.3.1). The PM based part of VCM is called Vertex Merging (VM), and the part derived from BPT is called Vertex Connection (VC). Thus, the full name: Vertex Connection and Merging. The names VM and VC refer to the way the connection between the light and camera sub-paths is done. The method of making the connection in the VM part of VCM is the main source of bias in the algorithm. Qin and others [QSH+15a] proposed an alternative, unbiased way for Vertex Merging called Unbiased Photon Gathering (UPG). The key finding of UPG is unbiased way to compute the probability density function of the merging event. The merging step is the VM equivalent for the BPT connection between the light and camera sub-paths. In the rest of the thesis we will use the term UPG both for the VCM algorithm with the unbiased alternative of VM and for the unbiased merging step itself.

The core of VCM is, as in the case of BPT, the multiple importance sampling

[7] In the implementation A_{i-1} and a_{i-1} are stored in the previous vertex, $\overleftarrow{g}^{\bar{y}}_{i-1}$ is computed using positions and normals of the current and previous vertices, for $\overleftarrow{q}^{\bar{y}}_{i-2}$ see the footnote at page 23.

estimator:

$$F_{VCM} = \frac{1}{n}\sum_{i=1}^{n}(C_{VC} + C_{VM}), \tag{2.53}$$

where C_{VC} and C_{VM} are defined as:

$$\begin{aligned} C_{VC} &= \frac{1}{n_{VC}}\sum_{j=0}^{n_{VC}}\sum_{s\geq 0,t\geq 1} w_{VC}(\bar{x}_{s,t})\frac{f_{VM}(\bar{x}_{s,t})}{p_{VC}(\bar{x}_{s,t})} = \frac{1}{n_{VC}}\sum_{j=0}^{n_{VC}}\sum_{s\geq 0,t\geq 1} w_{VC,s,t}\frac{f_{VC,s,t}}{p_{VC,s,t}} \\ C_{VM} &= \frac{1}{n_{VM}}\sum_{j=0}^{n_{VM}}\sum_{s\geq 2,t\geq 2} w_{VM}(\bar{x}_{s,t})\frac{f_{VM}(\bar{x}_{s,t})}{p_{VM}(\bar{x}_{s,t})} = \frac{1}{n_{VM}}\sum_{j=0}^{n_{VM}}\sum_{s\geq 2,t\geq 2} w_{VM,s,t}\frac{f_{VM,s,t}}{p_{VM,s,t}}. \end{aligned} \tag{2.54}$$

The estimator consists of two parts the VC part C_{VC} (which is essentially the BPT estimator) and the VM part C_{VM}. Every iteration (which corresponds to the sum in Equation 2.53) of VCM has two stages. In the first stage a list of n_{VM} light sub-paths $\bar{Y} = \{\bar{y}_{s_0}, \bar{y}_{s_1}, \ldots, \bar{y}_{s_{n_{VM}-1}}\}$ is generated and their vertices are stored in the acceleration structure (described in detail in Section 2.4.9). The acceleration structure have to support fixed-radius nearest neighbor queries. The radius r of the query is fixed for the single iteration of the algorithm and it is known at the time the acceleration structure is constructed. The number of VC samples v_{VC} is typically set to 1 and the number of VM samples (which corresponds to the number of traced light sub-paths) is set to the number of pixels of the rendered image [GKDS12].

The second stage is similar to tracing of the camera sub-paths in BPT. For every pixel of the light sensor a camera sub-path $\bar{z}_t$ is traced. Before the tracing begins, a single light sub-path from the set of sub-paths generated in the first stage is selected (usually it is $\bar{y}_{s_i}$, where i is the number of the pixel corresponding to the current camera sub-path[8]). While the camera sub-path is traced, for each of its vertices two steps are executed: the connection and the merging.

The connection is accomplished by linking the currently processed vertex of the camera sub-path $\bar{z}_t$ with each of the vertices of the light sub-path $\bar{y}_{s_i}$ selected beforehand and accumulating the contributions using the estimator C_{VC}. The factors for VC, that is $f_{VC,s,t}$ and $p_{VC,s,t}$, are exactly the same as their BPT equivalents $f_{s,t}$ and $p_{s,t}$ respectively (see Sections 2.3.3 and 2.3.4). Other than that, the are two differences between this algorithm and BPT. The first one is all of the light sub-paths are generated at the beginning of the iteration. The second difference is in the way the multiple importance weights are computed. The BPT connections are not the only ones contributing to the end result. The multiple importance sampling weights have to take the contributions of the VM paths into account as well. The details are in the following sections.

There are two ways the vertex merging can be done (see Figure 2.10). We will call them the merging from the camera perspective and the merging from the light

[8] In other words the light sub-paths are paired with camera sub-paths one-to-one in the order of their generation.

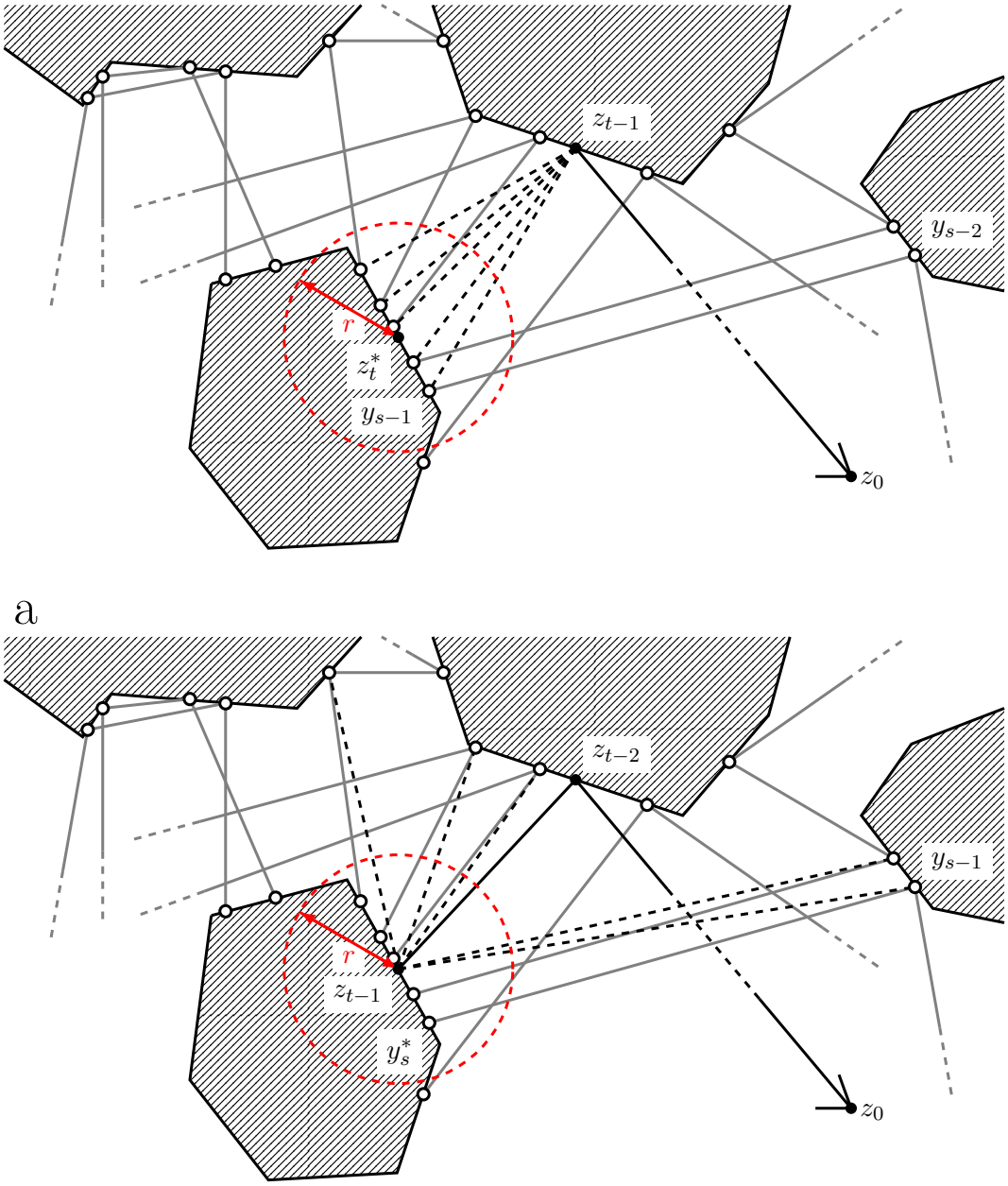

Figure 2.10: The first of two possible methods for the vertex merging: a) from the perspective of camera, b) from the perspective of light. The dashed lines are actual connections. The light vertices are marked with empty circles. The gray lines are the light sub-paths. For clarity only the vertices of single light sub-path were labeled. The red dashed circle represent gathering radius.

perspective [9]. The merging from the camera perspective is slightly simpler. It starts

[9] Qin and others [QSH+15a] proposed a heuristic which uses one or another method depending on the reflexivity of the surface. Though their heuristic is better that any of both methods alone, it greatly complicates already complex implementation of MIS weights computation and will not

from tracing of the tentative vertex z_t^* of the camera sub-path. The tentative vertex z_t^* will become the current vertex of the sub-path for the next iteration of the camera sub-path tracing. Before that happens, however, the acceleration structure is queried for all the light vertices in the proximity of r from the position of the tentative vertex. We will call this step a vertex gathering[10]. Notice, that the gathered vertices are the endpoints of the light sub-paths produced in the first stage of the algorithm. Now, the current vertex z_{t-1} (which is the vertex preceding the tentative vertex) of the camera sub-path is connected with each of the gathered endpoints of light sub-paths yielding set of full light camera paths. The contribution of every path $y_0 y_1 \dots y_{s_i-1} z_{t-1} \dots z_0$ is accumulated using the estimator C_{VM}. The tentative vertex becomes the next current vertex and the process repeats until Russian Roulette terminates it.

In the case of the merging from the light perspective the vertex gathering is done from the position of current camera vertex z_{t-1}. The current vertex z_{t-1} is connected with each of the vertices y_{s_i-1} preceding the gathered light vertices $y_{s_i}^*$. As such, we can think about the each of the gathered vertices as of the tentative vertices at the light sub-path. The final paths looks the same $y_0 y_1 \dots y_{s_i-1} z_{t-1} \dots z_0$, however it's important to remember that now y_{s_i-1} isn't the gathered one and the point of the gathering z_{t-1} is included in the path.

2.4.2 Acceptance Probability

If we consider the set of all possible connections between the light sub-paths and camera sub-paths the vertex gathering determines if given connection (and the path created as the result of it) is accepted or not. The path is accepted if and only if the endpoints of its light sub-path and camera sub-path are closer to each other than r. As such, the vertex gathering step of the vertex merging can be viewed as a probabilistic event with its own probability density function. We will call it the path acceptance probability.

This observation allows us to integrate the vertex gathering into the multiple importance sampling framework. The acceptance probability for the path $\bar{x}_{s,t}$ is the probability of the tentative vertex z_t^* of the camera sub-path $\bar{z}_t$ being sampled in the gathering neighborhood of the endpoint y_{s-1} of the light sub-path $\bar{y}_s$. The gathering neighborhood $\mathcal{N}(x)$ of the vertex v is the set of all the points belonging to the surfaces of the scene in the proximity r from the vertex v:

$$\mathcal{N}(v) = \{u \in \mathcal{M} \mid \|u - v\| \leq r\}. \tag{2.55}$$

With the gathering neighborhood defined above, the acceptance probability for the

be discussed in detail here.

[10]The vertex gathering step corresponds to the photon gathering in PM.

merging from the camera perspective can be written as:

$$p^{\bar{x},C}_{acc,s,t} = p_{acc}(\bar{x}_{s,t}) = P(\|y_{s-1} - z^*_t\| < r) = \int_{\mathcal{N}(y_{s-1})} p(z_{t-1} \to z) \cdot V(z_{t-1} \leftrightarrow z)\,\mathrm{d}z, \tag{2.56}$$

where $p(z_{t-1} \to z)$ is the probability of sampling the vertex z given the vertex z_{t-1} with respect to the area measure (see Section 2.3.4 and Figure 2.9 for the explanation of the notation). The visibility function $V(z_{t-1} \leftrightarrow z)$ (see Section 2.2.1) is necessary as the tentative ray may be overshadowed by the geometry and the tentative vertex z^*_t may not land in the gathering neighborhood even if sampled in its direction. For the merging from the light perspective the equation becomes:

$$p^{\bar{x},L}_{acc,s,t} = p_{acc}(\bar{x}_{s,t}) = P(\|z_{t-1} - y^*_s\| < r) = \int_{\mathcal{N}(z_{t-1})} p(y_{s-1} \to y) \cdot V(y_{s-1} \leftrightarrow y)\,\mathrm{d}y. \tag{2.57}$$

The final probability of sampling the path $\bar{x}_{s,t} = \bar{y}_s\bar{z}_t$ in VM is the product of the acceptance probability and the probabilities of sampling the light and camera subpaths (the indices C and L were dropped as the further derivations are the same for both cases):

$$p^{\bar{x}}_{VM,s,t} = p^{\bar{y}}_s p^{\bar{z}}_t p^{\bar{x}}_{acc,s,t}. \tag{2.58}$$

The direct evaluation of the above formula isn't possible, as there is no closed solution for the acceptance probability. The following approximation for the acceptance probability was proposed in [BfI03] (see Section 2.3.2 for the definition of the forward factor $\vec{p}^{\bar{z}}_t$)[11]:

$$p^{\bar{x}}_{acc,s,t} = \vec{p}^{\bar{y}}_s \pi r^2 = p(z_{s-1} \to y^*_s)\pi r^2. \tag{2.59}$$

This approximation uses reasonable assumption that the gathering neighborhood is usually very small and may be approximated it with a circle. The probability of sampling the tentative vertex is considered to be constant as well. This approximation is used in VCM and the error it introduces is the main source of the bias in the algorithm. However, the simplicity of the formula makes it very attractive from the computational efficiency point of view. To summarize, the probability density of sampling the whole path using VM and approximated acceptance probability is:

$$p^{\bar{x}}_{VM,s,t} = p^{\bar{y}}_s p^{\bar{z}}_t \vec{p}^{\bar{y}}_s \pi r^2. \tag{2.60}$$

2.4.3 Unbiased Acceptance Probability

Another approach to the problem of evaluation of the acceptance probability was presented in [QSH+15a]. It turns out, that the acceptance probability itself can be estimated in an unbiased fashion using the Monte Carlo methods. The approach is based on the observation that a repetitive generation of the tentative vertices is a Bernoulli process, whose success probability is equal to the acceptance probability.

[11] The equation for the camera perspective case is analogous.

The geometric distribution associated with the process gives the number $N(\bar{x}_{s,t})$ of trials required before the first success:

$$P(N(\bar{x}_{s,t}) = i) = p^{\bar{x}}_{acc,s,t} \cdot (1 - p^{\bar{x}}_{acc,s,t})^{i-1}. \tag{2.61}$$

The expected value of the number of the trials before the first success is equal to the inverse of the acceptance probability (Equation 2.62). It means that we can estimate the reciprocal of the probability density function for the vertex gathering simply by repeatedly sampling the tentative vertices.

$$E[N(\bar{x}_{s,t})] = \sum_{i=1}^{\infty} i \cdot p^{\bar{x}}_{acc,s,t} \cdot (1 - p^{\bar{x}}_{acc,s,t})^{i-1} = \frac{1}{p^{\bar{x}}_{acc,s,t}}. \tag{2.62}$$

The above scheme can't be used directly due to the high computational cost. The number of Bernoulli trials is potentially unbounded. Consider the situation in which the gathering neighborhood is far away from sampling point (see Figure 2.11). The expected number of required trials is going to be prohibitively high.

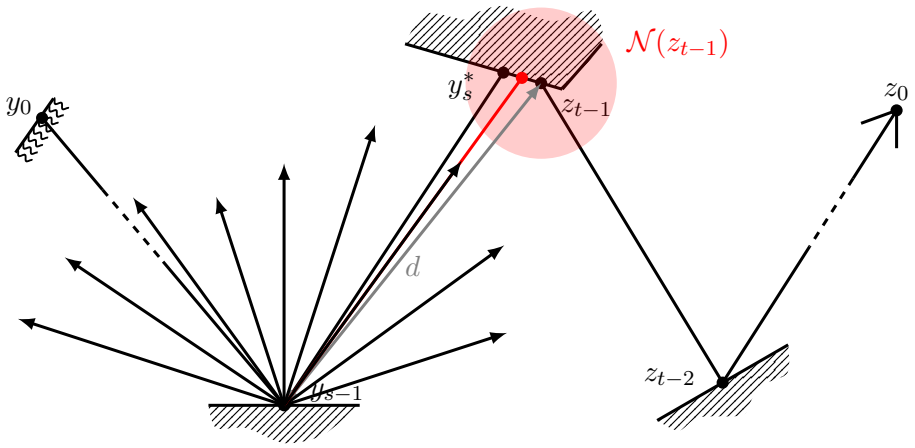

Figure 2.11: When $d >> r$ (where r is the radius of gathering, not marked for brevity) the probability of hitting the gathering neighborhood $\mathcal{N}(z_{t-1})$ (the intersection of the gathering circle and the surface) is prohibitively small.

In practice it isn't uncommon for the gathering radius to be many times smaller than the distance from the gathering neighborhood to the sampling point. It is important to mention that every trial requires a single test for the intersection between the tentative ray and the geometry of the scene, and the intersection tests are usually the most expensive operations in the ray tracing based algorithms.

To solve the problem Qin and others [QSH+15a] proposed an idea of limiting the number of Bernoulli trials which obviously wouldn't hit the gathering neighborhood. Let's recall that the tentative vertex is generated by tracing a ray from the position of the current end of the camera sub-path z_{t-1} and in the direction sampled from the distribution associated with the BSDF at that surface point. The sampling domain is the whole unit sphere $\mathcal{S}^2$ centered at z_{t-1}. The domain can be limited by projecting

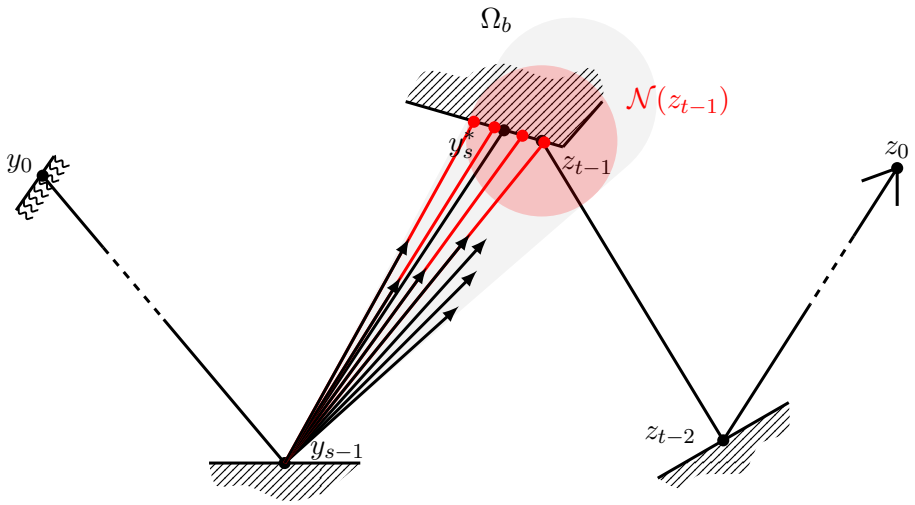

Figure 2.12: With the angular bound the tentative rays are focused in the gathering neighborhood $\mathcal{N}(z_{t-1})$. Notice however, that still some rays doesn't hit the neighborhood due to geometry of the surface.

the gathering neighborhood onto the unit sphere and introducing an angular bound. In this new scheme, the tentative rays are sampled from the original domain limited with this bound. The angular bound can be selected arbitrarily, however the tighter fit to the gathering neighborhood the better performance. Imposing the angular bound on the full sphere domain yields a new bounded domain Ω_b. The following equation presents the relationship between the expected value of the new geometric distribution on number of bounded trials $N^b(\bar{x}_{s,t})$ and the acceptance probability reciprocal:

$$\frac{1}{p^{\bar{x}}_{acc,s,t}} = \frac{E[N^b(\bar{x}_{s,t})]}{p^b_{s,t}}, \tag{2.63}$$

where $p^b_{s,t}$ is the adjustment factor defined as:

$$p^b_{s,t} = \int_{\Omega_b} p(y_{s-1} \to y)\,\mathrm{d}y. \tag{2.64}$$

The adjustment factor $p^b_{s,t}$ has to have a closed form solution. This requirement obviously limits the range of BSDF probability density functions for which the angular bound can be applied. The supplementary material [QSH+15a] for [QSH+15b] contains derivations of the angular bounds for Lambertian and Phong's BSDFs. To summarize, we can estimate the reciprocal of the acceptance probability by tracing $N^b(\bar{x}_{s,t})$ tentative rays, from the bounded sampling domain, until one of the tentative vertices lands in the gathering neighborhood and applying the normalization factor $p^b_{s,t}$ to the result:

$$\frac{1}{p^{\bar{x}}_{acc,s,t}} \approx \frac{N^b(\bar{x}_{s,t})}{p^b_{s,t}}. \tag{2.65}$$

Assuming that the distance to the gathering neighborhood $d = \|z_{t-1} - y_{s-1}\|$ is way greater than its radius r, the factor by which the number of Bernoulli trials

is reduced equals to:

$$\frac{E[N^b(\bar{x}_{s,t})]}{E[N(\bar{x}_{s,t})]} \approx \frac{\pi r^2}{d^2}. \tag{2.66}$$

The improvement is huge in practice, as for the most of the connections r is much smaller ($r << d$) than d. Qin gives the number 2.81 as the average number of trials per connection for practical scenes and it agrees with the results obtained by the author of this thesis. Nonetheless, the estimation of the gathering probability stays one of the main bottlenecks in the UPG implementation.

2.4.4 Bias in Measurement Contribution Function

The acceptance probability density function isn't the only source of bias in the VCM. The measurement contribution function can be evaluated in two ways as well. In unbiased case, the measurement contribution function is evaluated as:

$$f_{s,t}^{\bar{x}} = f_s^{\bar{y}} \cdot f_s(y_{s-2} \to y_{s-1} \to z_{t-1}) \cdot G(y_{s-1} \leftrightarrow z_{t-1}) \cdot f_s(y_{s-1} \to y_{t-1} \to z_{t-2}) \cdot f_t^{\bar{z}}. \tag{2.67}$$

Important here is the fact that the correct BSDF $f_s(y_{s-1} \to y_{t-1} \to z_{t-2})$ is used. In contrast, the approximation used in the VCM implementation contains the BSDF $f_s(z_t^* \to y_{t-1} \to z_{t-2})$. Assuming that the distance between the points z_t^* and y_{s-1} is negligibly small, the BSDF $f_s(z_t^* \to y_{t-1} \to z_{t-2})$ can be good approximation for the correct one:

$$f_s(z_t^* \to y_{t-1} \to z_{t-2}) \approx f_s(y_{s-1} \to y_{t-1} \to z_{t-2}). \tag{2.68}$$

Similar considerations apply to the geometry factor, in VCM the approximate version $G(z_t^* \leftrightarrow z_{t-1})$ is used. The usage of approximate BSDF has several advantages. The approximate evaluation is a little more efficient in practical implementation as it avoids recomputing BSDF related values for the new direction. For example in UPG the geometry factor for the new direction has to be recomputed, which requires an additional intersection test for the visibility factor embedded in the geometry factor. However, the main reason the approximation is used in VCM is the precise way doesn't support perfectly specular surfaces. The perfectly specular BSDF can be sampled only unidirectionally, when the path is traced. Because of the delta distributions, the BSDF evaluated at the end vertices of the light and camera sub-paths for explicit connection is always equal zero. The approximation circumvents this by reusing the BSDF of the tentative rays.

This is the reason for which UPG doesn't support ideal reflections and refractions. On the other hand, VCM can suffer from severe errors caused by the approximation.

2.4.5 MIS Weights for VCM

The computation of the MIS weights for BPT is already quite complicated (see Section 2.3.5). The VM part of VCM makes things even more complex, as contributions both from the vertex connection and vertex merging need to be taken into consideration. The general expression for the power heuristic weight in the context of VCM is (as before we drop the β exponent for clarity):

$$w^{\bar{x}}_{v,s,t} = \frac{n_v p^{\bar{x}}_{v,s,t}}{n_{VC} \sum\limits_{s' \geq 0, t' \geq 1} p^{\bar{x}}_{VC,s',t'} + n_{VM} \sum\limits_{s' \geq 2, t' \geq 2} p^{\bar{x}}_{VM,s',t'}}, \tag{2.69}$$

where v is either VC or VM. The reason for which the eye sub-paths for VC are not considered was explained in the BPT section. The light sub-paths with the length shorter than 2 are skipped in VM, because VC alone samples them usually in a more efficient manner than VM (see [GKDS12]). As it was the case for BPT, the weights defined as in Equation 2.69 cannot be evaluated directly. To enable efficient computation they need to be rewritten in a recursive manner.

First, we need to transform the basic formula for the VCM power heuristic weight (Equation 2.69) to its reciprocal form:

$$\begin{aligned} w^{\bar{x}}_{v,s,t} &= \frac{n_v p^{\bar{x}}_{v,s,t}}{n_{VC} \sum\limits_{s' \geq 0, t' \geq 1} p^{\bar{x}}_{VC,s',t'} + n_{VM} \sum\limits_{s' \geq 2, t' \geq 2} p^{\bar{x}}_{VM,s',t'}} \\ &= \frac{n_v p^{\bar{x}}_{v,s,t}}{n_{VC} \sum\limits_{i=0}^{s+t-1} p^{\bar{x}}_{VC,i,s+t-i} + n_{VM} \sum\limits_{i=2}^{s+t-2} p^{\bar{x}}_{VM,i,s+t-i}} \\ &= \left(\frac{n_{VC}}{n_v} \sum_{i=0}^{s+t-1} \frac{p^{\bar{x}}_{VC,i,s+t-i}}{p^{\bar{x}}_{v,s,t}} + \frac{n_{VM}}{n_v} \sum_{i=2}^{s+t-2} \frac{p^{\bar{x}}_{VM,i,s+t-i}}{p^{\bar{x}}_{v,s,t}} \right)^{-1} = \left(w^{-\bar{x}}_{v,s,t} \right)^{-1}. \end{aligned} \tag{2.70}$$

Let's recall the relation between the probability density functions for VC and VM (first defined in Equations 2.44 and 2.60):

$$\begin{aligned} p^{\bar{x}}_{VC,s,t} &= p^{\bar{y}}_s p^{\bar{z}}_t = \prod_{i=0}^{s-1} \vec{p}^{\,\bar{y}}_i \prod_{i=0}^{t-1} \vec{p}^{\,\bar{z}}_i \\ p^{\bar{x}}_{VM,s,t} &= p^{\bar{y}}_s p^{\bar{z}}_t \cdot \vec{p}^{\,\bar{y}}_s \pi r^2 = \prod_{i=0}^{s-1} \vec{p}^{\,\bar{y}}_i \prod_{i=0}^{t-1} \vec{p}^{\,\bar{z}}_i \cdot \vec{p}^{\,\bar{y}}_s \pi r^2 \\ &= p^{\bar{x}}_{VC,s,t} \cdot \vec{p}^{\,\bar{y}}_s \pi r^2. \end{aligned} \tag{2.71}$$

The probability density for VM contains the approximation of the acceptance probability $\vec{p}^{\,\bar{y}}_s \pi r^2$. The inverse weight for VM (Equation 2.72) can be expressed in terms

of the inverse weight for VC (Equation 2.73):

$$\begin{aligned} w^{-\bar{x}}_{VM,s,t} &= \frac{n_{VC}}{n_{VM}} \sum_{i=0}^{s+t-1} \frac{p^{\bar{x}}_{VC,i,s+t-i}}{p^{\bar{x}}_{VM,s,t}} + \frac{n_{VM}}{n_{VM}} \sum_{i=2}^{s+t-2} \frac{p^{\bar{x}}_{VM,i,s+t-i}}{p^{\bar{x}}_{VM,s,t}} \\ &= \frac{1}{\vec{p}^{\bar{y}}_s \pi r^2} \frac{n_{VC}}{n_{VM}} \left(\frac{n_{VC}}{n_{VC}} \sum_{i=0}^{s+t-1} \frac{p^{\bar{x}}_{VC,i,s+t-i}}{p^{\bar{x}}_{VC,s,t}} + \frac{n_{VM}}{n_{VC}} \sum_{i=2}^{s+t-2} \frac{p^{\bar{x}}_{VM,i,s+t-i}}{p^{\bar{x}}_{VC,s,t}} \right) \\ &= \frac{1}{\vec{p}^{\bar{y}}_s \pi r^2} \frac{n_{VC}}{n_{VM}} w^{-\bar{x}}_{VC,s,t}, \end{aligned} \tag{2.72}$$

$$w^{-\bar{x}}_{VC,s,t} = \frac{n_{VC}}{n_{VC}} \sum_{i=0}^{s+t-1} \frac{p^{\bar{x}}_{VC,i,s+t-i}}{p^{\bar{x}}_{VC,s,t}} + \frac{n_{VM}}{n_{VC}} \sum_{i=2}^{s+t-2} \frac{p^{\bar{x}}_{VM,i,s+t-i}}{p^{\bar{x}}_{VC,s,t}}. \tag{2.73}$$

To simplify the notation, let's introduce an η variable for the fraction:

$$\eta = \frac{n_{VM} \pi r^2}{n_{VC}}. \tag{2.74}$$

To write the weight in a recursive form, we need to rearrange the equations into more suitable shape. The following set of equations shows required transformations (the equations are valid for $s \geq 2$ and $t \geq 2$, for other cases the second sum is equal 0, without this condition the second step is invalid):

$$\begin{aligned} w^{-\bar{x}}_{VC,s,t} &= \frac{n_{VC}}{n_{VC}} \sum_{i=0}^{s+t-1} \frac{p^{\bar{x}}_{VC,i,s+t-i}}{p^{\bar{x}}_{VC,s,t}} + \frac{n_{VM}}{n_{VC}} \sum_{i=2}^{s+t-2} \frac{p^{\bar{x}}_{VM,i,s+t-i}}{p^{\bar{x}}_{VC,s,t}} \\ &\overset{1}{=} \frac{n_{VC}}{n_{VC}} \sum_{i=0}^{s} \frac{p^{\bar{x}}_{VC,i,s+t-i}}{p^{\bar{x}}_{VC,s,t}} + \frac{n_{VC}}{n_{VC}} \sum_{i=1}^{t-1} \frac{p^{\bar{x}}_{VC,s+t-i,i}}{p^{\bar{x}}_{VC,s,t}} \\ &+ \frac{n_{VM}}{n_{VC}} \sum_{i=2}^{s} \frac{p^{\bar{x}}_{VM,i,s+t-i}}{p^{\bar{x}}_{VC,s,t}} + \frac{n_{VM}}{n_{VC}} \sum_{i=2}^{t-1} \frac{p^{\bar{x}}_{VM,s+t-i,i}}{p^{\bar{x}}_{VC,s,t}} \\ &\overset{2}{=} \sum_{i=0}^{s-1} \frac{\vec{p}^{\bar{y}}_0 \cdots \vec{p}^{\bar{y}}_{i-1} \overleftarrow{p}^{\bar{y}}_i \cdots \overleftarrow{p}^{\bar{y}}_{s-1} \vec{p}^{\bar{z}}_{t-1} \cdots \vec{p}^{\bar{z}}_0}{\vec{p}^{\bar{y}}_0 \cdots \vec{p}^{\bar{y}}_{s-1} \vec{p}^{\bar{z}}_{t-1} \cdots \vec{p}^{\bar{z}}_0} \\ &+ \eta \sum_{i=2}^{s-1} \frac{\vec{p}^{\bar{y}}_0 \cdots \vec{p}^{\bar{y}}_{i-1} \vec{p}^{\bar{y}}_i \overleftarrow{p}^{\bar{y}}_i \cdots \overleftarrow{p}^{\bar{y}}_{s-1} \vec{p}^{\bar{z}}_{t-1} \cdots \vec{p}^{\bar{z}}_0}{\vec{p}^{\bar{y}}_0 \cdots \vec{p}^{\bar{y}}_{s-1} \vec{p}^{\bar{z}}_{t-1} \cdots \vec{p}^{\bar{z}}_0} \\ &+ \sum_{i=1}^{t-1} \frac{\vec{p}^{\bar{y}}_0 \cdots \vec{p}^{\bar{y}}_{s-1} \overleftarrow{p}^{\bar{z}}_{t-1} \cdots \overleftarrow{p}^{\bar{z}}_i \vec{p}^{\bar{z}}_{i-1} \cdots \vec{p}^{\bar{z}}_0}{\vec{p}^{\bar{y}}_0 \cdots \vec{p}^{\bar{y}}_{s-1} \vec{p}^{\bar{z}}_{t-1} \cdots \vec{p}^{\bar{z}}_0} \\ &+ \eta \sum_{i=2}^{t-1} \frac{\vec{p}^{\bar{y}}_0 \cdots \vec{p}^{\bar{y}}_{s-1} \overleftarrow{p}^{\bar{z}}_{t-1} \cdots \overleftarrow{p}^{\bar{z}}_i \vec{p}^{\bar{z}}_{i-1} \cdots \vec{p}^{\bar{z}}_0}{\vec{p}^{\bar{y}}_0 \cdots \vec{p}^{\bar{y}}_{s-1} \vec{p}^{\bar{z}}_{t-1} \cdots \vec{p}^{\bar{z}}_0} \\ &+ \eta \vec{p}^{\bar{y}}_s + 1 \end{aligned} \tag{2.75}$$

$$
\begin{aligned}
w_{VC,s,t}^{-\bar{x}} &\overset{3}{=} \sum_{i=0}^{s-1} \frac{\overleftarrow{p}_i^{\bar{y}} \cdots \overleftarrow{p}_{s-1}^{\bar{y}}}{\vec{p}_i^{\bar{y}} \cdots \vec{p}_{s-1}^{\bar{y}}} + \eta \sum_{i=2}^{s-1} \vec{p}_i^{\bar{y}} \frac{\overleftarrow{p}_i^{\bar{y}} \cdots \overleftarrow{p}_{s-1}^{\bar{y}}}{\vec{p}_i^{\bar{y}} \cdots \vec{p}_{s-1}^{\bar{y}}} \\
&+ \sum_{i=1}^{t-1} \frac{\overleftarrow{p}_{t-1}^{\bar{z}} \cdots \overleftarrow{p}_i^{\bar{z}}}{\vec{p}_{t-1}^{\bar{z}} \cdots \vec{p}_i^{\bar{z}}} + \eta \sum_{i=2}^{t-1} \frac{\overleftarrow{p}_{t-1}^{\bar{z}} \cdots \overleftarrow{p}_i^{\bar{z}}}{\vec{p}_{t-1}^{\bar{z}} \cdots \vec{p}_i^{\bar{z}}} \overleftarrow{p}_{i-1}^{\bar{z}} \\
&+ \eta \vec{p}_s^{\bar{y}} + 1 \\
&\overset{4}{=} \sum_{i=0}^{s-1} \prod_{j=i}^{s-1} \frac{\overleftarrow{p}_j^{\bar{y}}}{\vec{p}_j^{\bar{y}}} + \eta \sum_{i=2}^{s-1} \vec{p}_i^{\bar{y}} \prod_{j=i}^{s-1} \frac{\overleftarrow{p}_j^{\bar{y}}}{\vec{p}_j^{\bar{y}}} \\
&+ \sum_{i=1}^{t-1} \prod_{j=i}^{t-1} \frac{\overleftarrow{p}_j^{\bar{z}}}{\vec{p}_j^{\bar{z}}} + \eta \sum_{i=2}^{t-1} \overleftarrow{p}_{i-1}^{\bar{z}} \prod_{j=i}^{t-1} \frac{\overleftarrow{p}_j^{\bar{z}}}{\vec{p}_j^{\bar{z}}} \\
&+ \eta \vec{p}_s^{\bar{y}} + 1.
\end{aligned}
\tag{2.76}
$$

Firstly, the sums corresponding to VC and VM techniques are split into sub-sums corresponding to the light and eye sub-paths (1). The probability distribution functions are expanded and the terms for $i = s$ are moved out from the light sub-path sums (2). In the next step the common factors are reduced in the joint probability density function fractions (3). At the end the products are rewritten in a more concise form (4). The result expression is ready to be rewritten in a recursive manner. The following formula shows the recursive form of Equation 2.76:

$$
\begin{aligned}
w_{VC,s,t}^{-\bar{x}} &= (A_{s-1}\overleftarrow{q}_{s-2}^{\bar{y}} + a_{s-1})\overleftarrow{g}_{s-1}^{\bar{y}}\overleftarrow{q}_{s-1}^{\bar{y}} + (C_{t-1}\overleftarrow{q}_{t-2}^{\bar{z}} + c_{t-1})\overleftarrow{g}_{t-1}^{\bar{z}}\overleftarrow{q}_{t-1}^{\bar{z}} \\
&+ \eta(B_{s-1}\overleftarrow{q}_{s-2}^{\bar{y}} + \vec{g}_{t-1}^{\bar{y}}\vec{q}_{t-1}^{\bar{y}}b_{s-1})\overleftarrow{g}_{s-1}^{\bar{y}}\overleftarrow{q}_{s-1}^{\bar{y}} + \eta(D_{t-1}\overleftarrow{q}_{t-2}^{\bar{z}} + \overleftarrow{g}_{t-2}^{\bar{z}}\overleftarrow{q}_{t-2}^{\bar{z}}d_{t-1})\overleftarrow{g}_{t-1}^{\bar{z}}\overleftarrow{q}_{t-1}^{\bar{z}} \\
&+ \eta\vec{g}_s^{\bar{y}}\vec{q}_s^{\bar{y}} + 1.
\end{aligned}
\tag{2.77}
$$

Each of four sub-sums can be expanded using the same scheme as in the case of BPT (see Equation 2.49). The subexpressions A_i, a_i, C_i and c_i are defined in Equation 2.51. The recursive formulas for the remaining recursive subexpressions are as follows:

$$
\begin{aligned}
b_i &= \begin{cases} 0 & \text{for } i \in \{0, 1\} \\ \dfrac{1}{\vec{g}_i^{\bar{z}}\vec{q}_i^{\bar{z}}} & \text{for } i > 1 \end{cases} \\
B_i &= \begin{cases} 0 & \text{for } i = 0 \\ (B_{i-1}\overleftarrow{q}_{i-2}^{\bar{y}} + \vec{g}_{i-1}^{\bar{y}}\vec{q}_{i-1}^{\bar{y}}b_{i-1})\dfrac{\overleftarrow{g}_{i-1}^{\bar{y}}}{\vec{g}_i^{\bar{z}}\vec{q}_i^{\bar{z}}} & \text{for } i > 0 \end{cases} \\
d_i &= \begin{cases} 0 & \text{for } i \in \{0, 1\} \\ \dfrac{1}{\vec{g}_i^{\bar{z}}\vec{q}_i^{\bar{z}}} & \text{for } i > 1 \end{cases} \\
D_i &= \begin{cases} 0 & \text{for } i = 0 \\ (D_{i-1}\overleftarrow{q}_{i-2}^{\bar{z}} + \overleftarrow{g}_{i-2}^{\bar{z}}\overleftarrow{q}_{i-2}^{\bar{z}}d_{i-1})\overleftarrow{g}_{i-1}^{\bar{z}}d_i & \text{for } i > 0 \end{cases}.
\end{aligned}
\tag{2.78}
$$

Notice that both the base case and the general cases D_i for 1 and 2 are equal to 0

(the same stands for B_i):

$$\begin{aligned} D_1 &= (D_0 \overleftarrow{q}^{\bar{z}}_{-1} + \overleftarrow{g}^{\bar{z}}_{-1} \overleftarrow{q}^{\bar{z}}_{-1} d_0) \overleftarrow{g}^{\bar{z}}_0 d_1 = (0 \cdot \overleftarrow{q}^{\bar{z}}_{-1} + \overleftarrow{g}^{\bar{z}}_{-1} \overleftarrow{q}^{\bar{z}}_{-1} \cdot 0) \overleftarrow{g}^{\bar{z}}_0 d_1 = 0 \\ D_2 &= (D_1 \overleftarrow{q}^{\bar{z}}_0 + \overleftarrow{g}^{\bar{z}}_0 \overleftarrow{q}^{\bar{z}}_0 d_1) \overleftarrow{g}^{\bar{z}}_1 d_2 = (0 \cdot \overleftarrow{q}^{\bar{z}}_0 + \overleftarrow{g}^{\bar{z}}_0 \overleftarrow{q}^{\bar{z}}_0 \cdot 0) \overleftarrow{g}^{\bar{z}}_1 d_2 = 0. \end{aligned} \tag{2.79}$$

2.4.6 MIS Weights for UPG

The formulas for MIS weights from the previous sections apply both to VCM and UPG. However as they are expressed in terms of the approximation of the acceptance probability, one may wonder if it will not introduce an error to otherwise unbiased UPG technique. The answer is it will not because as long as the conditions from Equation 2.10 for the MIS weights are maintained the multiple importance sampling will stay unbiased. However, a sufficiently large error between the real and approximated acceptance probability may lead to suboptimal weighting. The error is especially large when the distance between the current camera vertex is close to the gathering radius r. To workaround the problem Qin and others [QSH+15a] proposed to clamp the approximation of the acceptance probability to 1 (see Section 2.4.2):

$$p^{\bar{x}}_{acc,s,t} = \min(\vec{p}^{\bar{y}}_s \pi r^2, 1). \tag{2.80}$$

The idea is that the acceptance probability cannot be larger than 1, so in cases when the basic approximation is greater than 1 the clamped approximation is always better. They argue, as well, that such a clamping isn't required for VCM. The bias in VCM usually manifests itself as excessive smoothing of the details. However its often a desirable property of biased methods to trade the increase in low frequency error, usually unnoticeable for human observer, for the computational efficiency.

To incorporate clamping into multiple importance sampling framework the equations for the VM multiple importance sampling weights have to be slightly modified. The following equations present the new, clamped incarnation of the formulas from the previous section:

$$\begin{aligned} w^{-\bar{x}}_{VC,s,t} = {} & (A_{s-1} \overleftarrow{q}^{\bar{y}}_{s-2} + a_{s-1}) \overleftarrow{g}^{\bar{y}}_{s-1} \overleftarrow{q}^{\bar{y}}_{s-1} + (C_{t-1} \overleftarrow{q}^{\bar{z}}_{t-2} + c_{t-1}) \overleftarrow{g}^{\bar{z}}_{t-1} \overleftarrow{q}^{\bar{z}}_{t-1} \\ & + \frac{n_{VM}}{n_{VC}} (B_{s-1} \overleftarrow{q}^{\bar{y}}_{s-2} + \min(\vec{g}^{\bar{y}}_{s-1} \vec{q}^{\bar{y}}_{s-1} \pi r^2, 1) \cdot b_{s-1}) \overleftarrow{g}^{\bar{y}}_{s-1} \overleftarrow{q}^{\bar{y}}_{s-1} \\ & + \frac{n_{VM}}{n_{VC}} (D_{t-1} \overleftarrow{q}^{\bar{z}}_{t-2} + \min(\overleftarrow{g}^{\bar{y}}_{t-2} \overleftarrow{q}^{\bar{y}}_{t-2} \pi r^2, 1) \cdot d_{t-1}) \overleftarrow{g}^{\bar{z}}_{t-1} \overleftarrow{q}^{\bar{z}}_{t-1} \\ & + \frac{n_{VM}}{n_{VC}} \min(\vec{g}^{\bar{y}}_{s-1} \vec{q}^{\bar{y}}_{s-1} \pi r^2, 1) + 1. \end{aligned} \tag{2.81}$$

$$
\begin{aligned}
b_i &= \begin{cases} 0 & \text{for } i \in \{0,1\} \\ \dfrac{1}{\vec{g}_i^{\bar{z}} \vec{q}_i^{\bar{z}}} & \text{for } i > 1 \end{cases} \\
B_i &= \begin{cases} 0 & \text{for } i = 0 \\ (B_{i-1} \overleftarrow{q}_{i-2}^{\bar{y}} + \min(\vec{g}_{i-1}^{\bar{y}} \vec{q}_{i-1}^{\bar{y}} \pi r^2, 1) \cdot b_{i-1}) \overleftarrow{g}_{i-1}^{\bar{y}} b_i & \text{for } i > 0 \end{cases} \\
d_i &= \begin{cases} 0 & \text{for } i \in \{0,1\} \\ \dfrac{1}{\vec{g}_i^{\bar{z}} \vec{q}_i^{\bar{z}}} & \text{for } i > 1 \end{cases} \\
D_i &= \begin{cases} 0 & \text{for } i = 0 \\ (D_{i-1} \overleftarrow{q}_{i-2}^{\bar{z}} + \min(\overleftarrow{g}_{i-2}^{\bar{z}} \overleftarrow{q}_{i-2}^{\bar{z}} \pi r^2, 1) \cdot d_{i-1}) \overleftarrow{g}_{i-1}^{\bar{z}} d_i & \text{for } i > 0 \end{cases}
\end{aligned}
\tag{2.82}
$$

2.4.7 Bias vs Consistency Interlude

An algorithm computing the integral $\int f(x)\,\mathrm{d}x$ is called unbiased if its results are correct on average. Otherwise it is called biased. More precisely, unbiasedness means that the expected value of the underlying estimator F_N is equal to the real value of the integral being evaluated:

$$E\left[F_N\right] - \int f(x)\,\mathrm{d}x = 0. \tag{2.83}$$

The algorithm is consistent, if the error of the result approaches 0 as the number of samples N goes to the infinity:

$$\lim_{N\to\infty} F_N - \int f(x)\,\mathrm{d}x = 0. \tag{2.84}$$

In general those two properties aren't related to each other. The estimator can be consistent and biased and vice versa. The consistency guarantees that the algorithm converges to the correct solution. All the methods presented in this thesis are consistent ones. The advantage of unbiased algorithms is ability to bound the error, which manifests itself usually as high frequency noise coming from variance.

2.4.8 VCM Consistency

The approximations in the acceptance probability and the contribution measurement functions introduce bias into VCM algorithm. However, it can be made consistent by shrinking the gathering radius r in subsequent iterations (see [GKDS12]).

$$r_i = r_1 \sqrt{i^{\alpha-1}} \tag{2.85}$$

The r_i variable in above equation is the radius in the i-th iteration and r_1 is the initial radius. The parameter α is an user parameter and it should be in range $(0, 1)$.

The RMS convergence rate of the VCM technique is bounded by:

$$\sqrt{O(n^{-1}) + O(n^{2(\beta+1)(\alpha-1)})}, \tag{2.86}$$

where the second term under the root is the error coming from bias. The β parameter is the one from MIS power heuristic. From above, it follows that for $\alpha \leq \frac{2\beta+1}{2\beta+2}$ has the optimal RMS convergence rate of $O(\sqrt{n^{-1}})$, which is the same as BPT's convergence.

2.4.9 Acceleration Structure

Both VCM and UPG algorithms require an acceleration structure for performing fast fixed-radius queries of the light sub-path vertices. The build time and the query time of the acceleration structure are critical, since they are the second most repeated operations after the ray-scene intersections. One such an acceleration structure suitable for vertex merging based techniques is the hash grid. The hash grid supports n-dimensional queries, but as the actual implementation uses the three dimensional one, we will focus on this particular case in the following explanations. The described approach scales easily to higher dimensions. For simplicity the figures show two dimensional version.

The basis of the hash grid data structure is a grid of cells with fixed size r. Every cell contains a set of points, which in our case represents the light vertices. The position f_{cell} of the cell for the point $p = [p_x, p_y, p_z]$ can be found with following formula:

$$f_{cell}(p) = \left[\left\lfloor \frac{p_x}{r} \right\rfloor, \left\lfloor \frac{p_y}{r} \right\rfloor, \left\lfloor \frac{p_z}{r} \right\rfloor \right]. \tag{2.87}$$

To find all the points within the distance r from the query point q, the cell $f_{cell}(q)$ corresponding to the query point q is found and the points from this cell and all of its 26 neighboring cells (total $27 = 3 \times 3 \times 3$ cells, see the green block of cells in Figure 2.14) are tested if they lie within the distance r to the query point.

This query scheme can be implemented with a hash map and a vector (an array) of points. The key for the hash map is the position of the cell and the value is a range of points from the vector. In the real implementation the range is defined as two integer indices: the index r_{begin} to the first point in the vector and the index r_{end} to the point next to the last one (so that number of points r_{size} in the range can be obtained with simple subtraction $r_{size} = r_{end} - r_{begin}$). For clarity [12] we will use the pair $[r_{begin}, r_{size}]$ to depict the range. The point ranges in the vector correspond to the cells from the hash grid, as presented in Figure 2.13.

To build the structure, the vector of points is sorted lexicographically (first by the z coordinate, then by y and by x at the end). Then, the consecutive ranges of points are grouped into the cells (thanks to the sorting the points from a single

[12] To limit number of arrows in the diagrams.

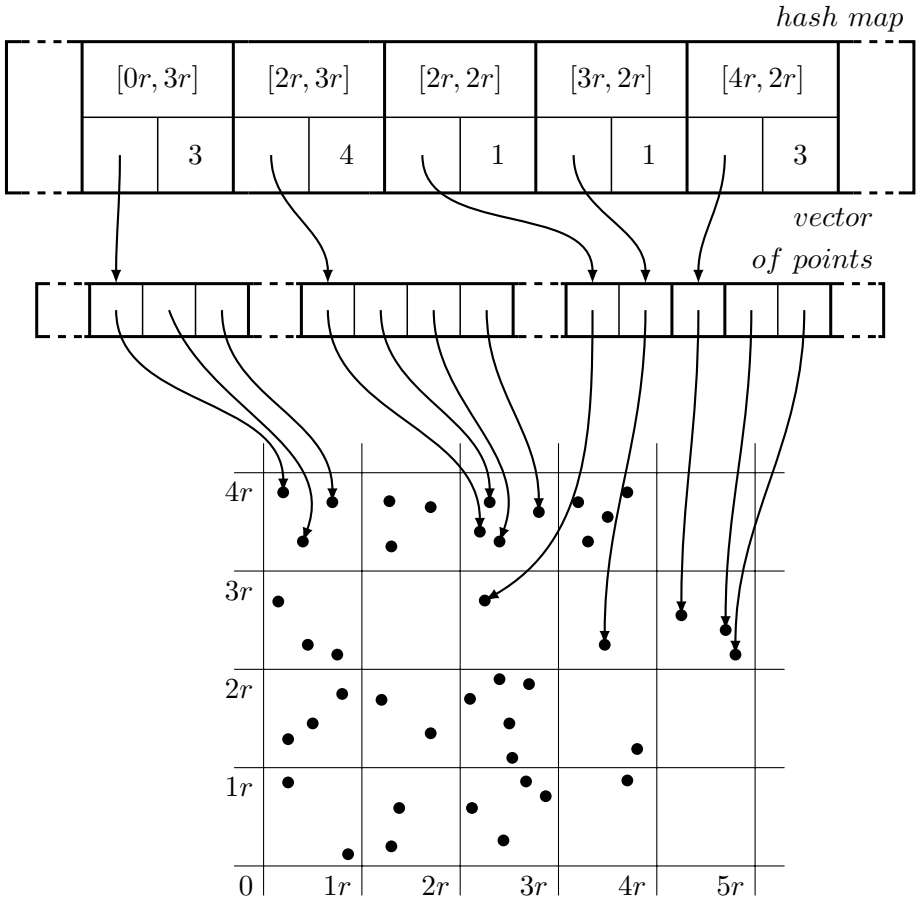

Figure 2.13: The overview of the two dimensional hash grid structure. The hash map stores the ranges of the points from the vector of the points. The ranges correspond to contents of certain cells.

cell occupy a continuous region of the vector). The hash map, which associates the ranges to the particular cells, is created at the end.

This basic build scheme can be improved. An observation can be made that not only the points inside a single cell, but the cells neighboring each other along the x axis, occupy continuous region inside the vector. We can exploit this fact and instead of storing in the hash map the range of points from the current cell only, we can store the range spanning over the neighbors of the current cell, see Figure 2.14. Several corner cases have to be handled here e.g. when one of the neighbors doesn't contain any points or the current cell is the last or the first cell in the row.

This improved approach has two advantages. The first one is that only 9 hash lookups, instead of 27, are required to visit all important cells. The second one is an improved cache locality and branch prediction, as the points from a single triplet of cells are traversed in one strike. The improvement in the performance of the hash map query, with respect to the basic building scheme, is almost 17% for the synthetic test case and over 50% for the more realistic bearings test case (see the

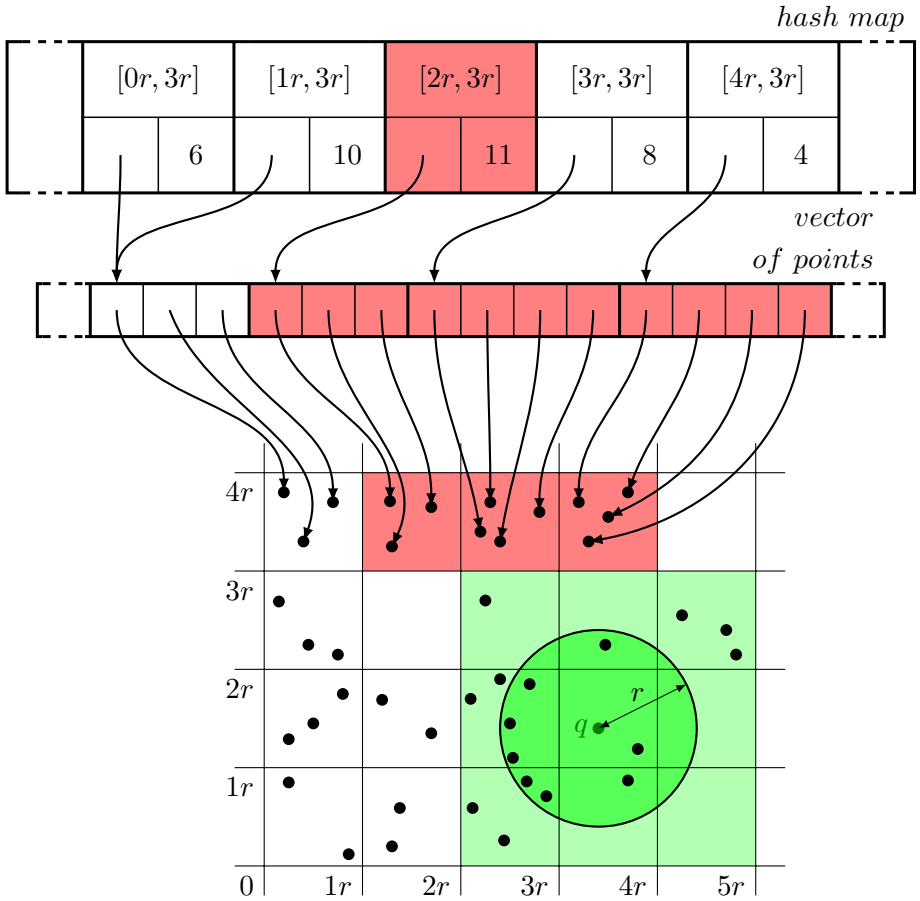

Figure 2.14: An improved layout of the hash grid. Storing the ranges of points from the triplets of cells saves on hash lookups when the structure is queried. The green part shows the query region with the center q and the radius r.

last paragraph in this section).

In the photon mapping algorithm the kd-tree is used for the queries. Since the kd-tree structure supports queries with an arbitrary radius size, it can be used for fixed radius queries as well. However, it turns out that the hash grid offers a better performance in practice.

For general d-dimensional kd-tree the complexity of the axis-parallel range query is $O(n^{1-\frac{1}{d}} + k)$ where k is number of reported points. The build complexity of both structures is similar $O(n \log n)$ [13].

To find out the algorithmic complexity of the hash grid we need to make some

[13] Technically the build procedure for the hash grid could be implemented without sorting, as $O(n)$ algorithm. However, then, it wouldn't be possible to directly store the points corresponding to particulars cells in continuous chunks of the memory (at least without additional rearrangements operations which would require sorting). It would prohibit the described optimization and in general have disastrous effect to the cache locality. In the implementation, for the typical n, the resulting constant would be much higher that the *logn* factor.

simplifications. Let's call the $3 \times 3 \times 3$ block of cells (the green cells in Figure 2.14) intersecting with the query region a query block. Let's assume that the points are distributed uniformly over the query block. The volume of the query block is equal to $v_{block} = (3r)^3 = 27r^3$ and the volume of the query region is equal to $v_{query} = \frac{4}{3}\pi r^3$. By the assumption of uniform distribution the number of reported points k is proportional to the number of points in the block n:

$$\frac{k}{n} \approx \frac{v_{query}}{v_{block}} = c. \tag{2.88}$$

The complexity of the query is linear $O(n)$ with respect to the number n of visited points in the block, and by above equation to the number of reported points $O(k)$. The assumption about the uniform distribution of the points inside the query block in in the context of vertex merging based techniques may look like a radical simplification. The light vertices are scattered randomly through the **surfaces** of the scene and not freely over the whole volume. Inside of the single query block they will be distributed uniformly over some planar surface (or multiple surfaces) intersecting with the query block. It means that the $O(k)$ is only a rough approximation. Nevertheless, as it can be seen in Figure 2.16 this bound is very close to the reality.

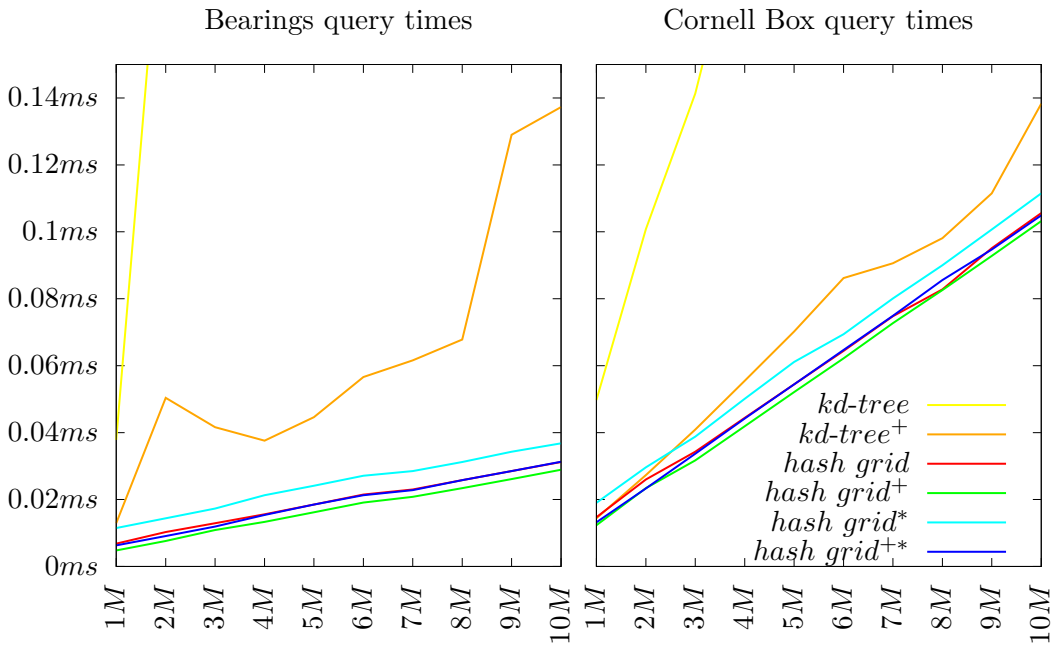

Figure 2.15: The query times with respect to the number of points in the scene. The M suffix stands for 10^6. The explanation of $\cdot^+$ and $\cdot^*$ can be found in Table 2.1. The charts shows that hash grid scales better than kd-tree as the number of points increases. The number of reported points k increases linearly with the number of points n in the scene (the horizontal axis) as the distribution of points is uniform, which confirms $O(k)$ complexity of hash grid query (see Figure 2.16 as well.

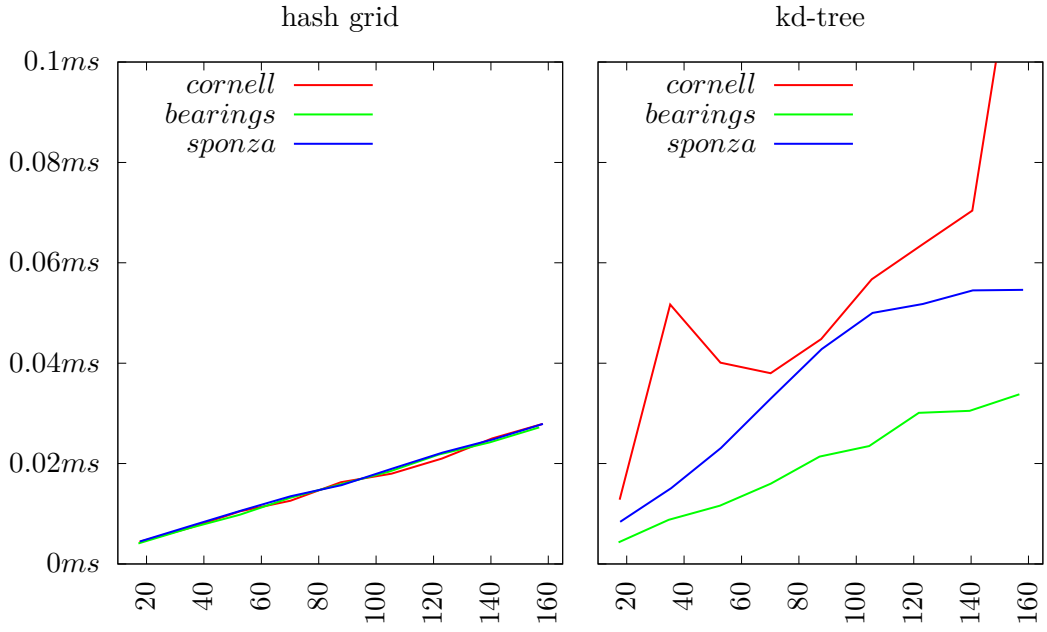

Figure 2.16: The time per query with respect to the average number of points k returned by the query. The relationship is linear and almost independent from the tested scene. The linear relationship confirms that hash grid query has a $O(k)$ query complexity.

The algorithms were tested on three different scenes: Cornell Box, Bearings and Sponza. Two sets of the points were uniformly scattered over the surfaces of the scene. The kd-tree and hash grid were constructed over the first set of queried points, then multiple fixed radius queries were executed at the positions from the second set. The build time and query times are presented in Table 2.1 and Figure 2.15. The hash-grid offers a considerably better build performance and query performance. The performance of the hash grid degrades in comparison to the kd-tree as the radius increases, however for the typical small radii the advantage of the hash grid is significant. The improvement in the build time for the optimized hash grid with respect to the optimized kd-tree is 36% - 67% (for $r = 0.01$). The improvement in the query time is 14% - 73%.

Structure	Build time $r = 0.01$	Query time $r = 0.01$	Build time $r = 0.1$	Query time $r = 0.1$
Bearings				
kd-tree	2.1418s	0.0497ms	2.7595s	2.8665ms
kd-tree$^+$	1.5503s	0.0144ms	1.9351s	0.7152ms
hash grid	0.6664s	0.0147ms	0.4207s	0.9058ms
hash grid$^+$	0.9831s	0.0123ms	0.4140s	0.9089ms
hash grid*	0.7170s	0.0190ms	0.4325s	0.9218ms
hash grid^{+*}	1.4518s	0.0132ms	0.4443s	0.9122ms
Cornell Box				
kd-tree	2.1951s	0.0378ms	2.0284s	1.0029ms
kd-tree$^+$	1.6130s	0.0129ms	1.5598s	0.2619ms
hash grid	0.4631s	0.0068ms	0.3726s	0.2985ms
hash grid$^+$	0.5217s	0.0048ms	0.3654s	0.2968ms
hash grid*	0.4649s	0.0115ms	0.3676s	0.3026ms
hash grid^{+*}	0.5557s	0.0063ms	0.3731s	0.2978ms
Sponza				
kd-tree	2.0638s	0.0179ms	2.0652s	0.2992ms
kd-tree$^+$	1.8117s	0.0067ms	1.5713s	0.0860ms
hash grid	0.6440s	0.0038ms	0.4551s	0.0841ms
hash grid$^+$	0.9488s	0.0018ms	0.4633s	0.0819ms
hash grid*	0.7314s	0.0095ms	0.4613s	0.0897ms
hash grid^{+*}	1.5294s	0.0035ms	0.5238s	0.0842ms

Table 2.1: The build and query times of different acceleration structures. The grayed out rows show the times for the micro-optimized versions of structures, marked with $\cdot^+$ as well. The green cells mark the best time in the column. The tests were conducted for two different query radii $r = 0.01$ and $r = 0.1$. The versions using slower `std::unordered_map` were marked with $\cdot^*$.

Chapter 3

Implementation

3.1 Overview

An integral part of this thesis is an implementation of PT, BPT, VCM and UPG. The implementation is written purely in software (no GPU acceleration) using C++. Intel's Embree[1] library was used as a ray tracing back-end (the library facilitates an API for creation of the acceleration structure and querying for intersections between rays and scene primitives). The advanced SIMD features of Embree, which allow to trace packets of rays simultaneously, weren't used. The reason was a lack of time to optimize all four techniques to use them. Nevertheless, the author appreciates the fact that it would be very interesting to see the influence of SIMD on the performance of probability reciprocal estimation in UPG, what is its current performance bottleneck.

Thanks to Assimp[2] library the implemented program is able to directly load .blend Blender files (and reload them on modification), what greatly accelerated the implementation-testing cycle. The glad[3], glfw[4] and imgui[5] libraries were used to create an interactive, real time preview of the rendering progress. The glm[6] library provided basic mathematical primitives like 3– and 4–dimensional vectors, matrices and operations on them. The `ska::flat_hash_map`[7] library was used as high performance drop-in replacement for `std::unordered_map` in the hash grid implementation. The OpenEXR[8] library was used to save the result images in high dynamic range .exr format.

Despite the fact that some of the techniques can be considered as the gener-

[1] `https://embree.github.io`
[2] `https://github.com/assimp/assimp`
[3] `https://github.com/Dav1dde/glad`
[4] `http://glfw.org`
[5] `https://github.com/ocornut/imgui`
[6] `https://glm.g-truc.net`
[7] `https://github.com/skarupke/flat_hash_map`
[8] `https://openexr.com`

alization of the others (e.g. BPT as generalization of PT), the PT and BPT were implemented separately from VCM and UPG. It allowed to cross check the correctness of the implementation by comparing the results from independent code paths.

3.2 Building and Running

The implementation can be found in a public git repository under the address `https://github.com/ciechowoj/master`. The git repository is mostly self contained (the dependencies are provided as git submodules), however a few external dependencies need to be installed on Linux system to be able to build the program. The list of required libraries is available in `README.md` file in the root directory of the repository. If all the dependencies are fulfilled the implementation can be build simply with the `make` command. The program can be compiled and run under Microsoft Windows using Visual Studio, however the build process isn't as straightforward as on Linux[9].

The initial build may take a significant amount of time as it builds the submodules from scratch. After the build process is completed the main executable file `master.bin` can be found in the `build/master` subdirectory. For convenience we will simply call it `master`. The primary way to use the `master` program is a command line interface. The full set of possible parameters can be found in Appendix A.

3.3 Rendering Scenes

The main functionality of the program is rendering of the 3d scenes. The scenes have to be in the .blend format (created with Blender software). Obviously the program supports only very limited set of features offered by Blender. Only the basic mesh models with normal vectors are supported (there is no support for additional per vertex data like texture coordinates). Figure 3.1 shows adjustable parameters. For the camera the *focal length* and *sensor size* can be modified. Multiple cameras are supported (selected with the `--camera` parameter). For lights the color and *energy* can be specified (the resultant light power is the color multiplied by the *power*). If the *diffuse* flag is selected the light is an area light, otherwise it is a directional light. The *diffuse* and *specular* colors are the coefficients of the energy conserving Phong BSDF (k_d and k_s respectively, in Equation 3.1), the *hardness* is a value of shiness exponent (the n coefficient, for more details see Lafortune and Willems paper [LW94b]). The α variable is an angle between the perfect specular reflection and the outgoing direction. The factors k_d and k_s have to obey the inequality $k_d + k_s \leq 1$ otherwise the BSDF will not conserve energy.

$$f_r(x, \omega_i, \omega_o) = k_d \frac{1}{\pi} + k_s \frac{n+2}{2\pi} \cos^n \alpha \tag{3.1}$$

[9] One complication is it requires manual building of some dependencies.

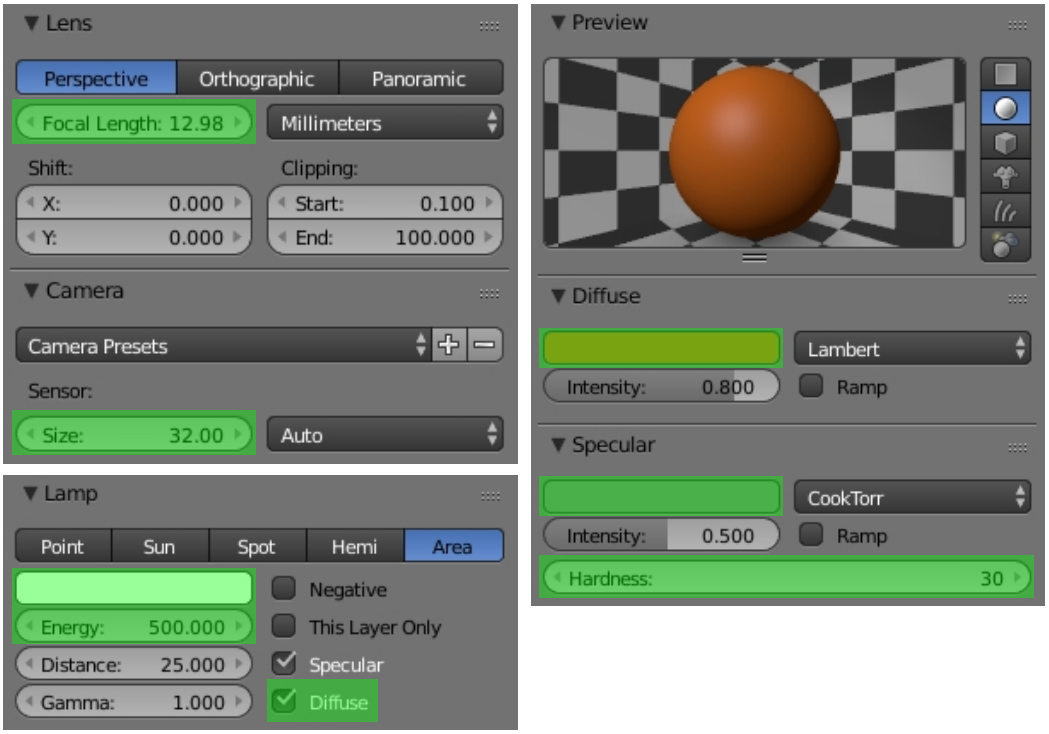

Figure 3.1: The adjustable parameters (highlighted with green color) of camera, materials and light respectively seen in the Blender interface.

The output of the program is a high dynamic range image in the .exr format. The high dynamic range images cannot be directly represented on paper or displayed on contemporary low dynamic range monitors. The images need to be tone mapped first, that is converted to low dynamic range. The wide support for .exr format and the existence of open source software for tone mapping with support to .ext files were the reasons to use this format. The freely available `luminance-hdr` software was used to tone map the images presented in this thesis. The `master` program can also be used as a simple .exr viewer but its tone mapping capabilities are limited to simple linear scaling, which is fine for a quick preview but unsuitable for a presentation. The linearly remapped images are either overexposed or the detail in shadows is lost.

The output files from `master` cannot be readily opened by `luminance-hdr` tool. The raw .exr result from `master` contains a special s channel (beside r, g, b channels) which represents number of samples generated for the particular pixel. The idea is to decrease the accumulation of floating point errors by delaying the division by the number of samples until the very end of rendering. To convert the raw .exr output from `master` to the standard .exr file, the `master bake` command has to be used. It makes the division by the number of samples and removes the extra s channel from the .exr file.

3.4 Auxiliary Tools

The `master` program provides several tools to manage the long rendering times and a huge number of result images with different combinations of rendering parameters.

The `master` command accepts a special `--reference` parameter. The argument of this parameter is a path to the reference image. The reference image is used to compute the errors during rendering. For every sample the absolute error and the root mean square errors are computed and their values are stored for later analysis. In order to store the error measurement the `master` program exploits another feature of the .exr format, which is support for custom meta data. The information about errors is saved inside the file with the image itself. This is very convenient, because otherwise the data would have to be stored externally doubling the number of output files to manage. The information about errors can be extracted from .exr meta data using `master measurements` command.

By default the errors are computed for the whole image. The `--trace` parameter allows to specify additional rectangular fragments of the image for which the errors should be computed.

The meta data from the image can be stripped with the `master strip` command. The `master merge` command allows to merge the results from multiple files into a single one (e.g. the rendering can be done on multiple machines or in multiple runs). The `master continue` command allows to restart the rendering in the place where it was stopped (it is handy in case of failure or when the machine is needed for something else). The `--snapshot` parameter enables the auto save feature. The parameter accepts a number of minutes as its argument. When it is specified the output file will be overwritten periodically with the current result of rendering. The `master diff` command computes relative difference between the input images, for an example, see the next section (Section 3.5).

3.5 Implementation Correctness

The implementation turned out to be a very challenging process. It was relatively easy to get visually convincing results, however getting precise, physically correct images is a different story. To facilitate this the implementation was tested on the number of test scenes[10].

The simplest test scene is called a furnace test and it consists of a closed volume with certain properties. The surfaces which make up the volume boundaries are covered with diffuse reflectors and emitters having a constant reflectance $\rho_d = \pi f_d$ and emittance L_e (the factor f_d is a constant value of the diffuse BSDF). The radiance L at any point of such a scene is constant and its value can be calculated

[10]The scenes can be found in `models` subdirectory of the repository.

analytically. The value of the radiance is equal to:

$$L = \sum_{i=0}^{\infty} \rho_d^i L_e = \frac{L_e}{1 - \rho_d}. \tag{3.2}$$

The above equation is just a decomposition of the total radiance into the radiances reaching some point of the scene after $n-1$ reflections (in other words: coming along the path of length n). The value of the radiance which reaches the point along the paths of length n is equal to:

$$L_r^{(n)}(\omega) = L_r^{(n)} = \rho_d^{(n-1)} L_e. \tag{3.3}$$

The meaning of this equation is the radiance $L_r^{(n)}$ along the path of length n is constant ($L_r^{(n)} = \rho_d^{(n-1)} L_e$) and independent from the direction ($L_r^{(n)}(\omega) = L_r^{(n)}$). It can be proven by simple induction over the path length n. For the paths with length 1, the lights are directly visible, so the radiance is equal to $L_r^{(1)} = L_e$. For the paths with length n, the intensity of reflected light $L_r^{(n)}$ is equal to:

$$\begin{aligned} L_r^{(n)} &= \int_{\mathcal{S}^2} f_d(\omega_o, \omega_i) L_r^{(n-1)}(\omega_i) |\omega_i \cdot n| \, \mathrm{d}\omega_i \\ &= \int_{\mathcal{S}^2} \frac{\rho_d}{\pi} L_r^{(n-1)} |\omega_i \cdot n| \, \mathrm{d}\omega_i \\ &= \frac{\rho_d}{\pi} L_r^{(n-1)} \int_{\mathcal{S}^2} |\omega_i \cdot n| \, \mathrm{d}\omega_i \\ &= \rho_d L_r^{(n-1)} = \rho_d^{(n-1)} L_e. \end{aligned} \tag{3.4}$$

By inductive assumption, the radiance $L_r^{(n-1)}(\omega_i)$ incoming over the paths of length $n-1$ is constant and directionally independent, thus we can move it out of the integral in the third transformation.

In our case[11] (see Figure 3.2) the scene consists of a cube and six diffuse area lights, aligned with the walls of the cube, emitting the radiance $L_e = \frac{1}{2}$. The cube is made of a diffuse material with reflectance $\rho_d = \pi f_d = \frac{1}{2}$. Thus, the radiance L reaching camera is equal to $\frac{L_e}{1-\rho_d} = 1$. This result can be readily used to verify the correctness of the output computed by the program.

In general the radiance reaching the camera cannot be computed analytically. For the other test cases[12] a high precision reference image was rendered using BPT and the results from other algorithms were compared against it[13]. The test scenes were designed to verify different configurations of lighting, geometry and materials or to pinpoint the root cause of a certain bug. For simplicity the lights in the scenes are calibrated, so that the average radiance reaching the camera is equal to 1 (in other words the average radiance calculated over all the pixels in the image is equal to 1). This allows to quickly determine if something is wrong without comparing the image with the reference one.

[11] `models/TestCaseFurnace.blend`

[12] `models/TestCase0.blend` - `models/TestCase18.blend`

[13] The correctness of BPT implementation was tested against the basic PT algorithm.

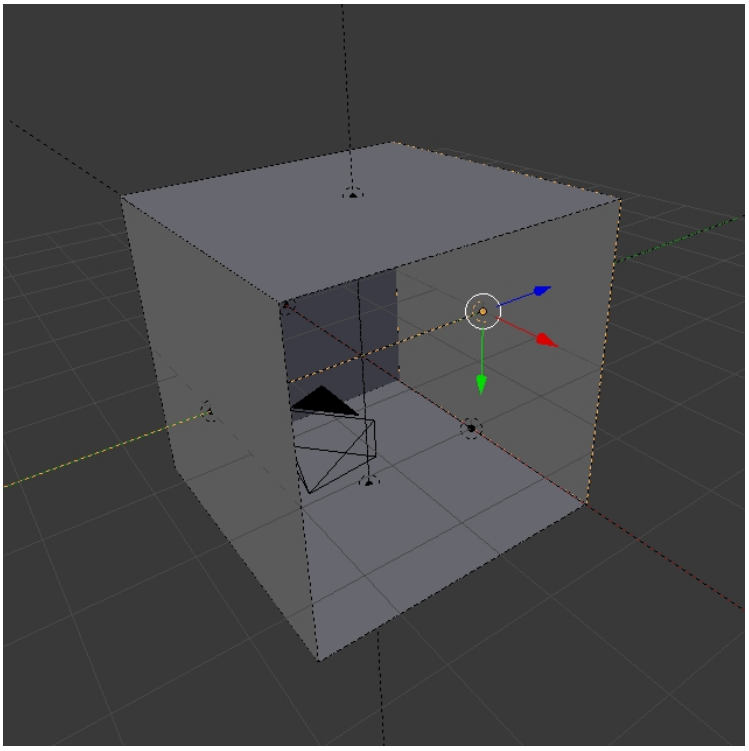

Figure 3.2: A screenshot from Blender presenting the furnace test scene. The front wall of the cube is removed to show the interior. The right area light (out of six) is selected.

Multiple issues were detected using the above approach with the test scenes. Some of the problems had nonobvious sources like poor quality of the random number generator, a bug in the acceleration structure or different kinds of problems with floating point numbers[14]. Despite described efforts to eliminate all the bugs, the implementation still shows error in some circumstances. In the bearings scene the BPT and UPG give slightly different results in the particular area of the scene (the space between the rings of the horizontal bearing). Figure 3.3 shows a closeup to the problematic area. The difference means that there have to be an error in the implementation of one of the techniques as both algorithms are unbiased. The possible sources of the difference include: numerical errors (e.g. caused by rising floating point numbers to high powers for highly reflective Phong material), inefficiency of BPT (in presence of highly reflective surfaces the probability of sampling long paths is extremely slow) or simply a bug in the implementation. The error may be present in other scenes as well, thus it may have an influence on the results presented at the end of this thesis. The author was unable to pinpoint the root cause of the problem so far. One difficulty in doing so, is the only scene among the tested ones where the phenomenon appears is the bearings scene. In particular, it doesn't show up in the simpler test scenes (like the test cases described at the beginning of the

[14]Floating point numbers can cause all sorts of problems. The traps range from precision loss and instability to things like different hashes for +0 and -0.

section). The very long rendering time required to catch such subtle errors poses a great difficulty for debugging. Usually, it takes only a few samples per pixel to see that something is wrong, however in the aforementioned case the error is so minute that it takes a few thousand of samples to decrease the noise to the level where the error starts to be visible. Those problems make the standard fix-build-verify approach inapplicable.

3.6 Performance Considerations

The implementation was profiled using the `gprof` tool. The Embree library provides efficient facilities to perform ray-triangle intersections, so the optimization effort was directed to other parts of the program. A considerable effort was put to optimize the fixed radius query acceleration structure — hash grid. Initial experiments showed that even a highly optimized kd-tree is slower in performing fast fixed radius queries than a simple hash grid data structure (see Section 2.4.9 for more details). Surprisingly, a great improvement in the speed was obtained by replacing the `std::unordered_map` with the highly optimized `ska::flat_hash_map`. Table 3.1 shows the best times for `std::unordered_map` and `ska::flat_hash_map` from multiple runs of the program compiled with one of the structures enabled[15]. The version with `ska::flat_hash_map` is 30% faster than the version with the hash

time	class name
4.51s	`ska::flat_hash_map`
5.87s	`std::unordered_map`

Table 3.1: `ska::flat_hash_map` vs `std::unordered_map`.

map from the C++ standard library. In addition to optimizing the data structures a bunch of micro-optimizations like devirtualization and inlining of critical functions (methods) was applied through the whole program. The computations for different parts of the image were parallelized. Table 2.1 presents the simplified output from `gprof` for the final version of the program[15].

As we can see the additional gains could be only acquired by optimizing the ray-triangle intersections itself (`occluded` and `intersect`). One approach to improve the speed of intersections would be to use the batch versions of the ray-triangle intersection routines which can trace a whole packet of multiple rays at once. It is expected that this would give additional boost in the speed, however for the price of increased complexity of the whole implementation. The complexity would be increased, as other parts of the implementation would need to handle the batches of

[15]The program was run with `master models/CrytekSponza.blend --UPG --parallel --num-samples=20 --radius=0.01 --batch --output=/tmp/test.exr`.

% time self	function name
23.16	`embree::avx::BVHNIntersector1<...>::occluded(...)`
19.46	`embree::avx::BVHNIntersector1<...>::intersect(...)`
8.91	`UPGBase<...>::_gather(...)`
7.00	`UPGBase<...>::_connect(...)`
4.88	`Scene::querySurface(...) const`
3.43	`DiffuseBSDF::query(...) const`
2.95	`Scene::occluded(...) const`
2.88	`ska::detailv3::sherwood_v3_table<...>::grow()`
2.02	`UPGBase<...>::_traceEye(...)`
2.00	`HashGrid3D<...>::build(...)`
1.94	`PhongBSDF::query(...) const`
1.15	`std::mersenne_twister_engine<...>::operator()()`
1.11	`std::sort<...>(...)`
1.06	`Scene::intersect(...) const`
18.05	other
% time total	function name
15.32	`UPGBase<...>::scatter(...)`

Table 3.2: The profiler output for the sample run of the program. The timings (*time self*) in the upper part of the table are exclusive (the function time doesn't include the time of the child if the child appears in the table as well). The lower part of the table contains the total timings (*time total*), that is the actual time that was spent inside the function.

rays.

Another potential improvement would be to increase the CPU utilization by parallelizing the construction process of the acceleration structure. The `scatter` function performs the generation of the light vertices and construction of the acceleration structure over them. The `scatter` function isn't currently parallelized, so that during its execution the CPU is underutilized. The improvement would be moderate, as currently it uses only 15% of the execution time (see Table 3.2).

Instead of tracing the light vertices at the beginning of each iteration we could use the light vertices from the previous iteration and in parallel to the main part of the program (the tracing of the eye sub-paths), generate the light vertices and build the acceleration structure for the next iteration. That way the `scatter` function could be run in parallel with the actual tracing of the eye paths, in which case the CPU would be fully utilized.

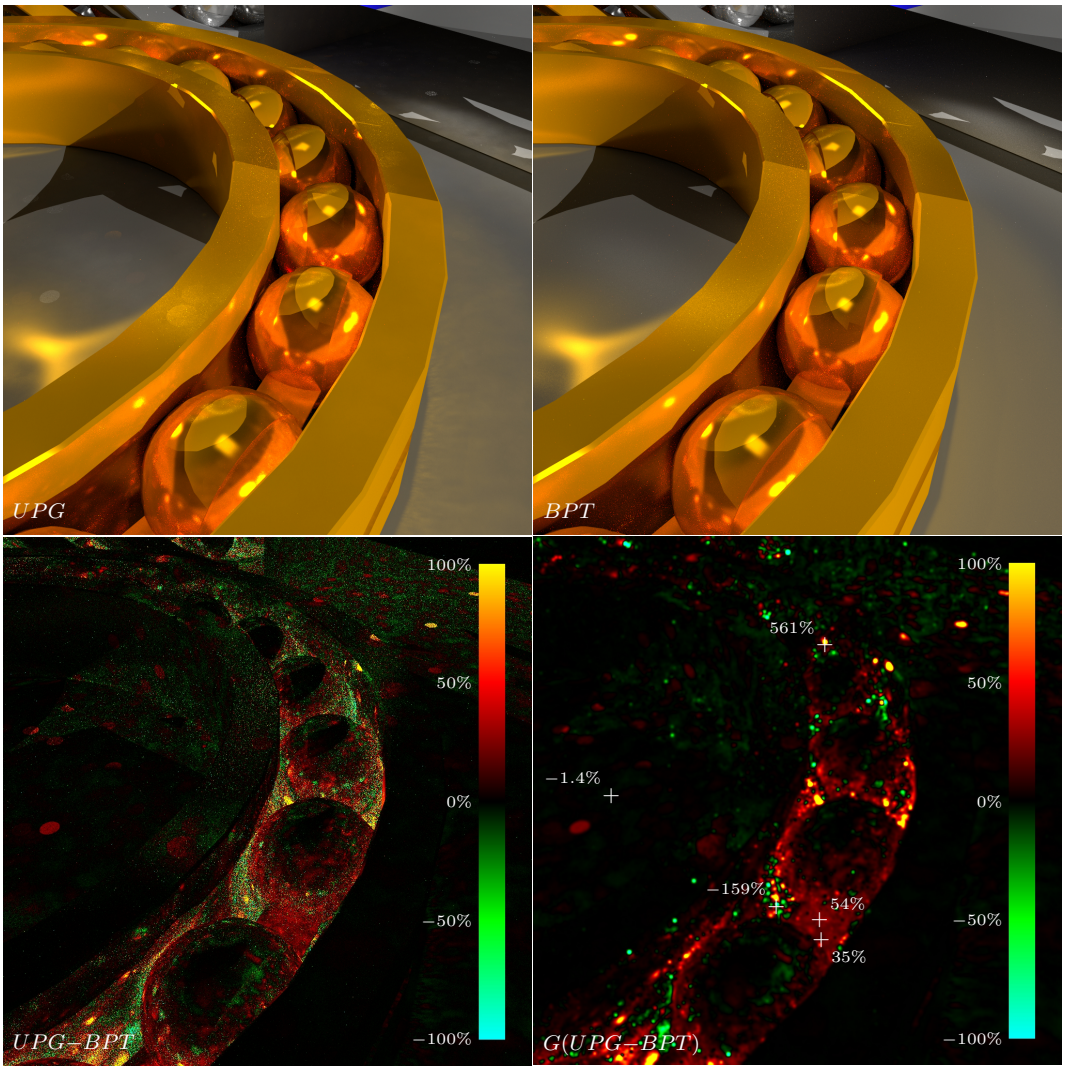

Figure 3.3: The closeup to the fragment of Bearings scene rendered using UPG and BPT. The bottom images present a relative difference $\frac{P_{UPG}-P_{BPT}}{\min(P_{UPG},P_{BPT})}$ between top images, where P is the brightness $\sqrt{P_r^2+P_g^2+P_b^2}$ of a pixel. The Gaussian blur was applied to the right bottom image to make it more clear. The region between the bearing rings is significantly brighter on the image rendered using UPG than on the image rendered using BPT which suggest some kind of bug in the implementation. The presence of the error in the area where the surfaces are very close to each other suggests some problems with floating points computations.

Chapter 4

Results

4.1 Test Scenes

The scenes Bathroom (Salle de Bain)[1], Breakfast Room[2] and Crytek Sponza[3] come from the Morgan McGuire's website [McG17]. The Bearings scene was modeled by the author of this thesis. The pocket watch[4] lying on the table in the Breakfast Room scene was created by the user *totopremier* from the website `blendswap.com`[5]. The geometry of the clock's face was added by the author (the original face was a texture, and texturing isn't supported by the program). The original materials and lights from all the scenes were discarded and a new arrangement was created by the author (the original materials and lights were incompatible with the program). The light arrangement is described in more detail under the figures with the result images (see Section 4.4).

4.2 Rendering of Results

The scenes were rendered using the BPT, UPG, UPG* (UPG with gathering from the camera) and VCM techniques. Firstly, for every scene the high quality image was rendered using BPT with a long rendering time (at least 48h). The reference images were rendered on a machine equipped with Intel Core i5-3317U CPU @ 1.70GHz and 8 GB RAM with a resolution of 1024×1024 pixels. Table 4.1 shows a summary of the rendering times for the particular scenes. Some of the scenes were rendered multiple times with different lighting or camera arrangements (the camera id is the number in the superscript of the scene name).

[1] Licence CC BY 3.0, Copyright Nacimus Ait Cherif

[2] Licence CC BY 3.0, Copyright Wig42

[3] Licence CC BY 3.0, Copyright 2010 Frank Meinl, Crytek

[4] Licence CC BY 3.0, Copyright totopremier

[5] As a matter of fact some other scenes from the McGuire's website originate from `blendswap.com` as well.

Scene	$Time$	$Samples$
$Bathroom$	$72h$	44082
$Bearings$	$263h$	834279
$BreakfastRoom1^0$	$58h$	35334
$BreakfastRoom1^1$	$56h$	28394
$BreakfastRoom2^0$	$58h$	30157
$BreakfastRoom2^1$	$54h$	26263
$CrytekSponza^0$	$88h$	89558
$CrytekSponza^1$	$71h$	68989
$CrytekSponza^2$	$101h$	83942

Table 4.1: The time of the rendering and the number of samples per pixel for the reference images.

Rendering algorithms are parametrized by super-parameters whose automatic adjustment is an open problem. In our case, they are the beta parameter β for multiple importance sampling (affecting all algorithms presented here), the gathering radius r (for vertex merging based techniques) and the alpha parameter α for a radius shrinking scheme (related to VCM). For the beta, the value 2 was chosen as suggested by Veach in his thesis [Vea97]. For the VCM alpha parameter the value of 0.75 was chosen (see [GKDS12] and Section 2.4.8). To find out the optimal value for the gathering radius, every scene was rendered multiple times with different values of the radius. The rendering time was at least 20 minutes. The detailed data from the rendering can be found in the table in Appendix B. The optimal choices of the radii are presented in Table 4.2. Figure 4.1 shows the influence of the gathering radius on the rendering error for the $BreakfastRoom2^1$ scene. The choice of the gathering radius is important as the difference in the error is significant for different radii. The range of reasonably good radii varies from scene to scene.

The result images were rendered using parameters as described above[6]. The machine equipped with Intel Xeon E5-2673 v3 @ 2.4GHz and 16 GB RAM was used for rendering. The rendering time of 6 hours was used and the resolution was set to the same as of the reference images (1024×1024).

4.3 Discussion

The performance of the rendering algorithm is determined by the speed with which the image converges to the reference image, that is the speed with which the absolute approximation error decreases to zero. For the test scenes the performance of

[6]The exact command line invocation can be found in the script `GenerateFinalResults.ps1` in the repository.

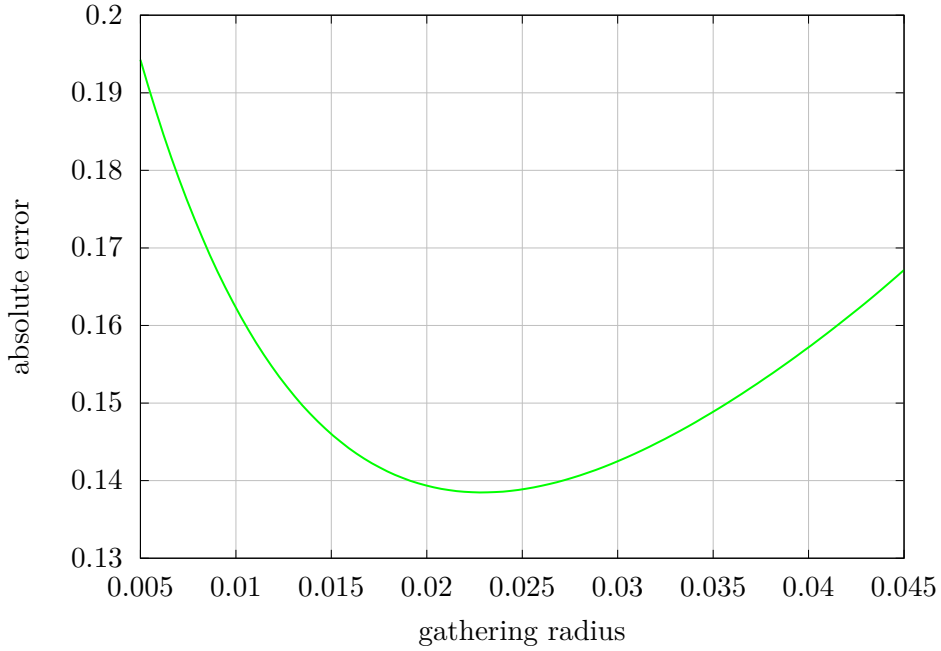

Figure 4.1: The correspondence between the gathering radius size and the absolute error for the $BreakfastRoom2$[1] scene rendered using UPG. Notice, that the y-axis doesn't start from zero. The optimal value is well pronounced.

the UPG and VCM algorithms ranges, from being on par with BPT, to being significantly better (consider the convergence charts in Figures 4.4-4.12). The *Bathroom* scene is the only scene, among the tested ones, where vertex merging based algorithms are slightly worse than classic BPT. The improvement in the performance is noticeable especially for the features which are considered hard to render using the classic path tracing algorithms (PT, BPT). The vertex merging based algorithms efficiently handle caustics induced by highly reflective surfaces (see the excerpts b, c and h from Figure 4.5 and the excerpt e from Figure 4.8) and are frequently superior, when it comes to rendering specular effects (see the excerpts a - e from Figure 4.8 and Figure 4.12), however it isn't a rule (e.g. the reflection from the excerpt c in Figure 4.9).

The advantage of the vertex merging based algorithms isn't limited only to the specular effects. The UPG and VCM are better in rendering the areas of the images illuminated by indirect light. The difference in efficiency grows with the number of reflections (see indirectly illuminated areas in Crytek Sponza and the interior of the bearing box in the Bearings scene). The $BreakfastRoom2$ (see Figure 4.8 and Figure 4.9) scene is the most striking example of this phenomenon. The images rendered with VCM and UPG are actually better than the reference image generated with BPT. Conversely, the common property of the scenes where the vertex based algorithms have worse performance is a big amount of direct lighting or lighting with a low number of reflections. In the Bathroom scene (Figure 4.4), where almost the

Scene	UPG	UPG^*	VCM
$Bathroom^0$	0.05	0.045	0.1
$Bearings^0$	0.03	0.02	0.04
$BreakfastRoom1^0$	0.01	0.01	0.025
$BreakfastRoom1^1$	0.01	0.01	0.05
$BreakfastRoom2^0$	0.015	0.01	0.015
$BreakfastRoom2^1$	0.02	0.01	0.05
$CrytekSponza^0$	0.02	0.01	0.035
$CrytekSponza^1$	0.02	0.01	0.035
$CrytekSponza^2$	0.02	0.01	0.03

Table 4.2: The optimal radii for particular techniques. The optimality criterion is an absolute error between the image in question and the reference image.

whole visible region is directly illuminated with the area light, the error in the images rendered by UPG and VCM is actually greater than in the image rendered with BPT. The advantage of BPT for direct lighting is visible in the $BreakfastRoom1$ scene too. Still, some areas in the shadows in this scene have better quality in the images rendered with UPG and VCM (e.g. excerpt *c*, Figure 4.9).

Another property of the vertex merging techniques where the merging is done from the light perspective (UPG) is a low frequency error. This kind of error tends to be less questionable for a human observer than the high frequency spikes generated by vanilla BPT (the characteristic "fireflies" noise) or the alternative version of UPG (UPG*). For example, compare the area around the jug in the images generated with BPT and UPG in Figure 4.9 or the light reflection on the first-plane ledge in Figure 4.11 (e.g. excerpt *c*).

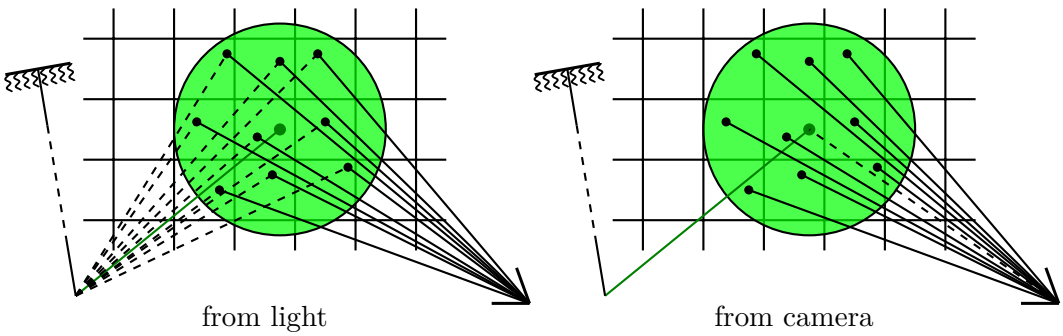

Figure 4.2: Two different variants of the vertex merging for the primary camera rays: from the light perspective (on the left) and from the camera perspective (on the right).

The low frequency noise in UPG is a result of a common light sub-path being shared by the multiple pixels which are in the close proximity to themselves. Let's

consider Figure 4.2. The grid represents the pixels of the image. For multiple pixels the same light vertex is gathered (the dark green one at the center of the gathering region), hence the value of multiple pixels is generated using the same light sub-path. This phenomenon can be observed as characteristic patches of pixels (see Figure 4.3) and results in low frequency noise.

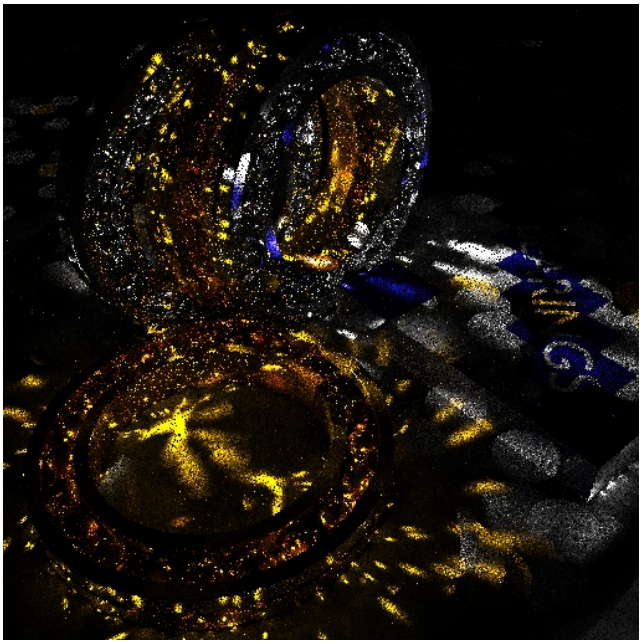

Figure 4.3: An image generated with UPG with single sample per pixel. The direct lighting is disabled for clarity. The bright patches correspond to the particular gathering areas.

This smoothing behavior, when beneficial in most cases, sometimes may cause problems. Consider a situation where a high intensity light sub-path is sampled. In case of BPT it would be connected only to one camera sub-path, which would result in the high frequency firefly pixel characteristic to BPT. However, in the case of UPG (and VCM) the light sub-path is merged to multiple camera sub-paths. This may produce ugly artifacts in the image (see the reflection in the floor in the Crytek Sponza scene, Figure 4.11). The artifacts will disappear as more samples are generated, but still they have potential to spoil otherwise high quality renderings.

The UPG* algorithm is a variant of UPG which does the vertex gathering from the camera perspective. When its efficiency for caustics and specular effects is still better than that of BPT, the general performance is worse than of the first variant. The reason for this is UPG* doesn't have the smoothing behavior of UPG. The primary camera rays are merged directly with the gathered light vertex (consider the right sub-figure in Figure 4.2)[7].

The biased version of UPG: the VCM algorithm produces images visually sim-

[7]This kind of merging is equivalent to the BPT connection. It is explicitly ignored in the implementation.

ilar to the ones generated by UPG. Unfortunately, the results generated by VCM suffer from bias artifacts, which are especially visible in the corners and other places where two surfaces are close to each other. The artifacts are severe enough to be noticed by a human even on the renderings with significant number of samples. The most spectacular examples of those artifacts can be seen in the excerpts $c-e$ in Figure 4.4, excerpt j in Figure 4.6, excerpt b in Figure 4.9 and excerpt j in Figure 4.12. Additionally, if the radius reduction scheme is used for VCM (which usually is, as it is required for consistency) VCM degenerates to BPT over time. It means that only the first samples of VCM benefit from the efficiency of the vertex merging and the performance will decrease as the number of samples increases.

4.4 Result Images

Below figures present the obtained results. The numbers in the corners of the zoomed excerpts are the RMS errors computed only for the excerpt areas. The source regions of the excerpts are marked with color coded rectangles and letters. The charts in the top right corner presents the relationship between the absolute error and the rendering time. The y-axes of the charts are logarithmic. The images generated with some techniques were omitted for brevity, the full images and their .exr versions can be found in the repository with the program.

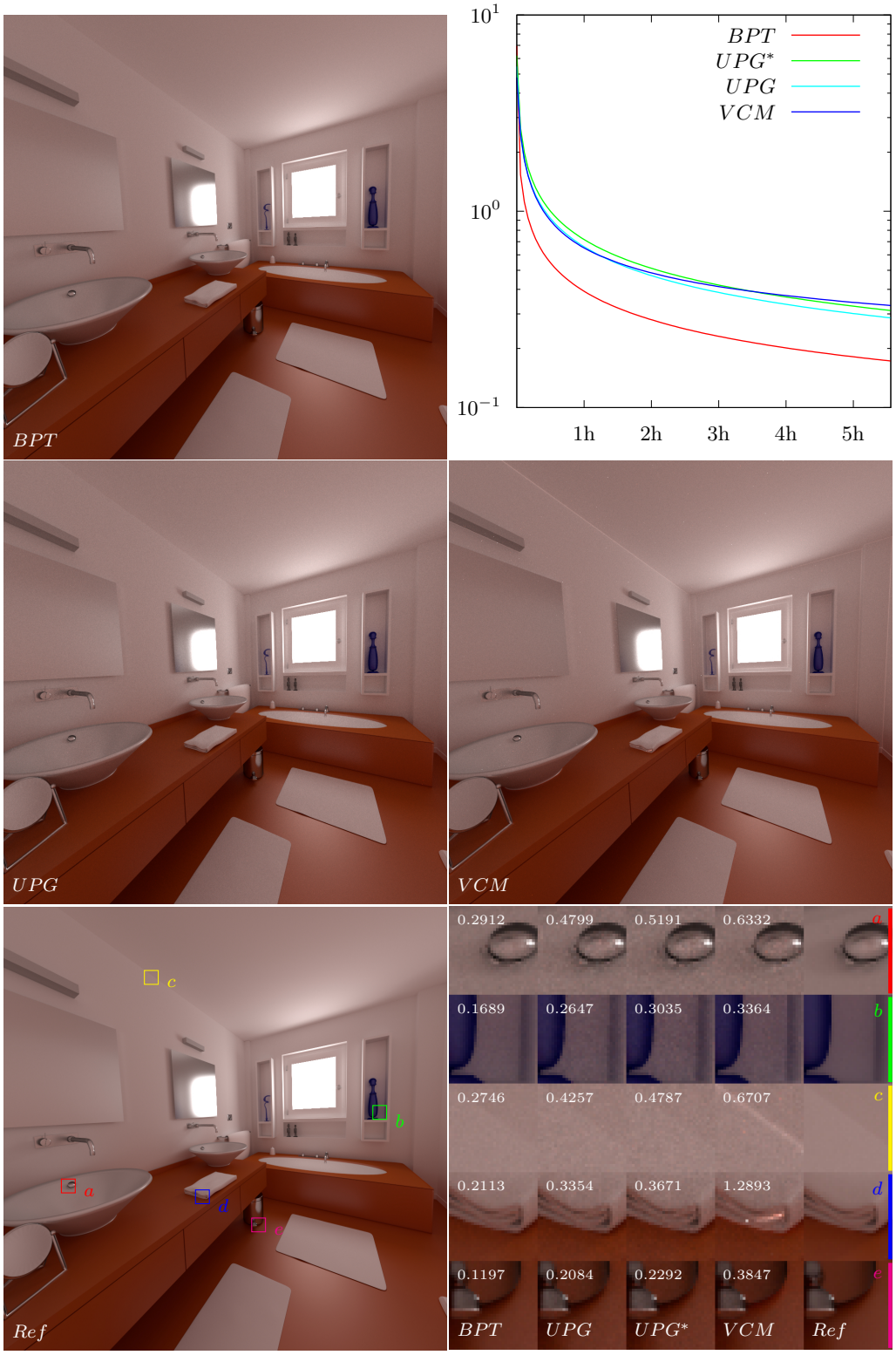

Figure 4.4: The *Bathroom* scene. The scene presents interior of a bathroom. The chrome elements and the mirrors are simulated using Phong material with a high shininess coefficient. The scene is illuminated by a single diffuse area light placed instead of the window glass. For the explanation of the markings see Section 4.4.

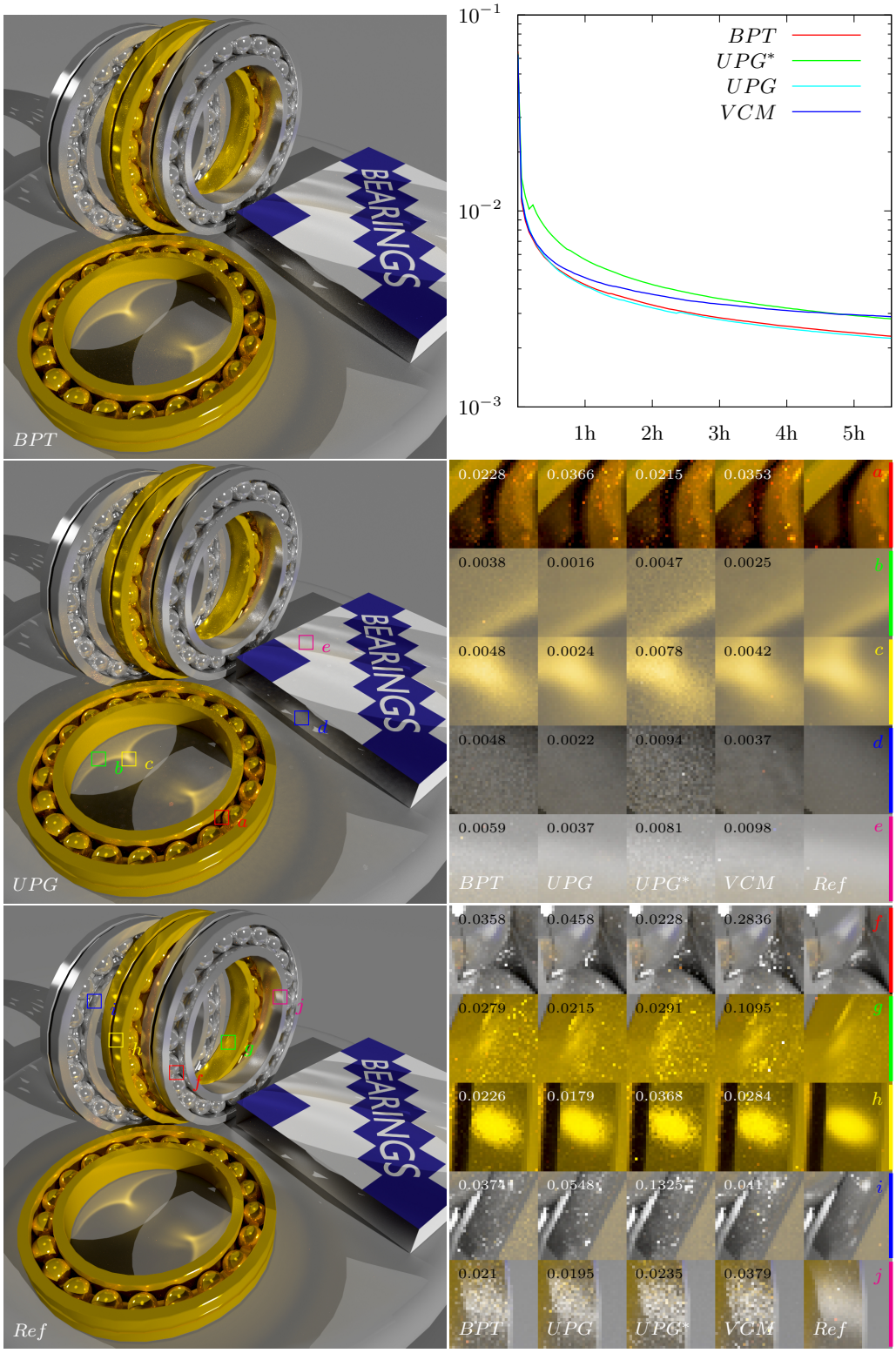

Figure 4.5: The *Bearings* scene. The scene presents ball bearings and a cardboard box for their packaging. The scene is illuminated by two directional lights (their arrangement can be guessed from the positions of cardioid caustics) and one weaker overhead diffuse area light. For the explanation of the markings see Section 4.4.

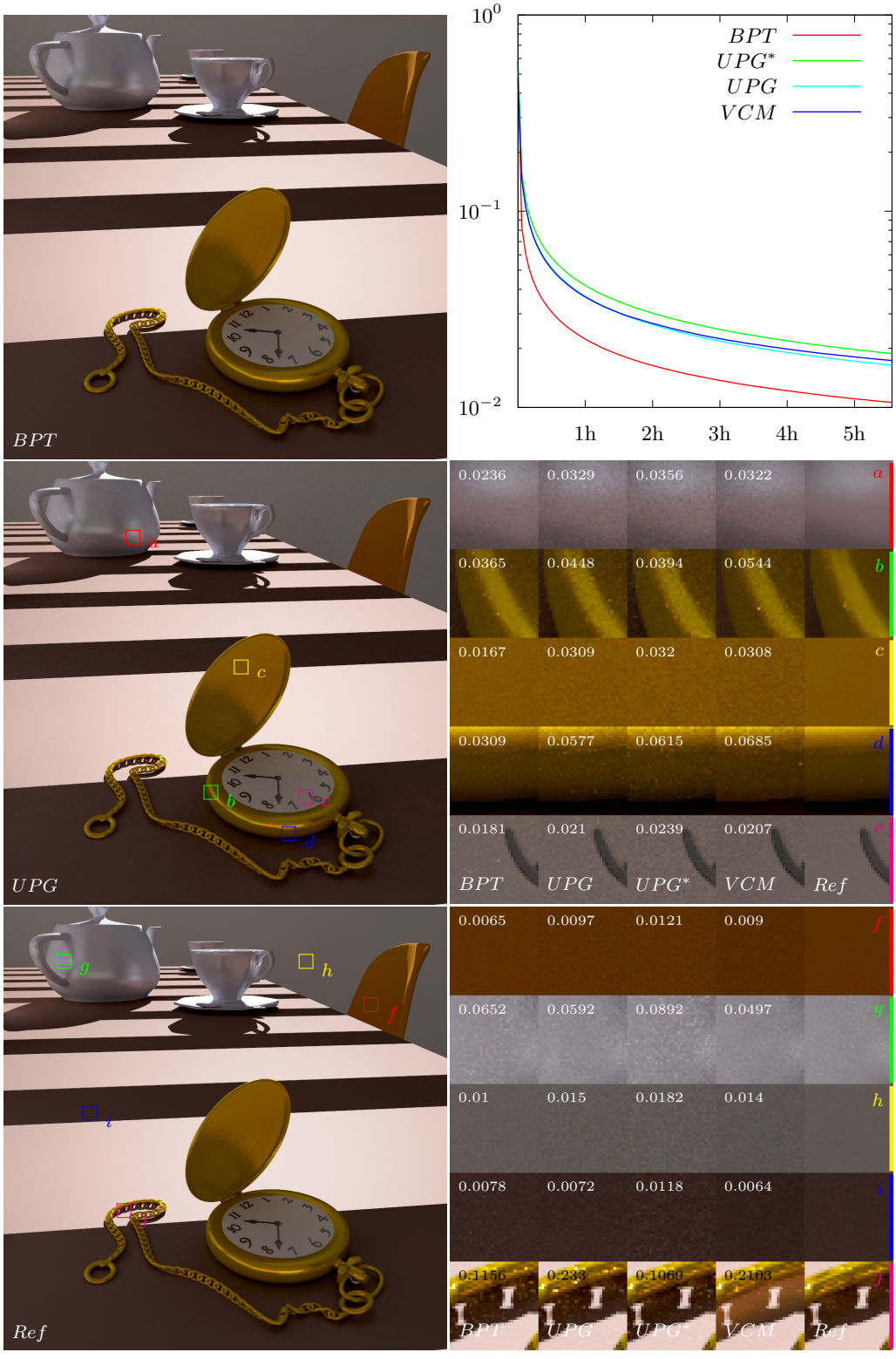

Figure 4.6: The $BreakfastRoom1^0$ scene. The scene presents a breakfast room illuminated by the sunlight passing through blinds. The sunlight is simulated with single directional light. Pay attention to the severe artifacts under the chain for VCM rendering (excerpt j). For the explanation of the markings see Section 4.4.

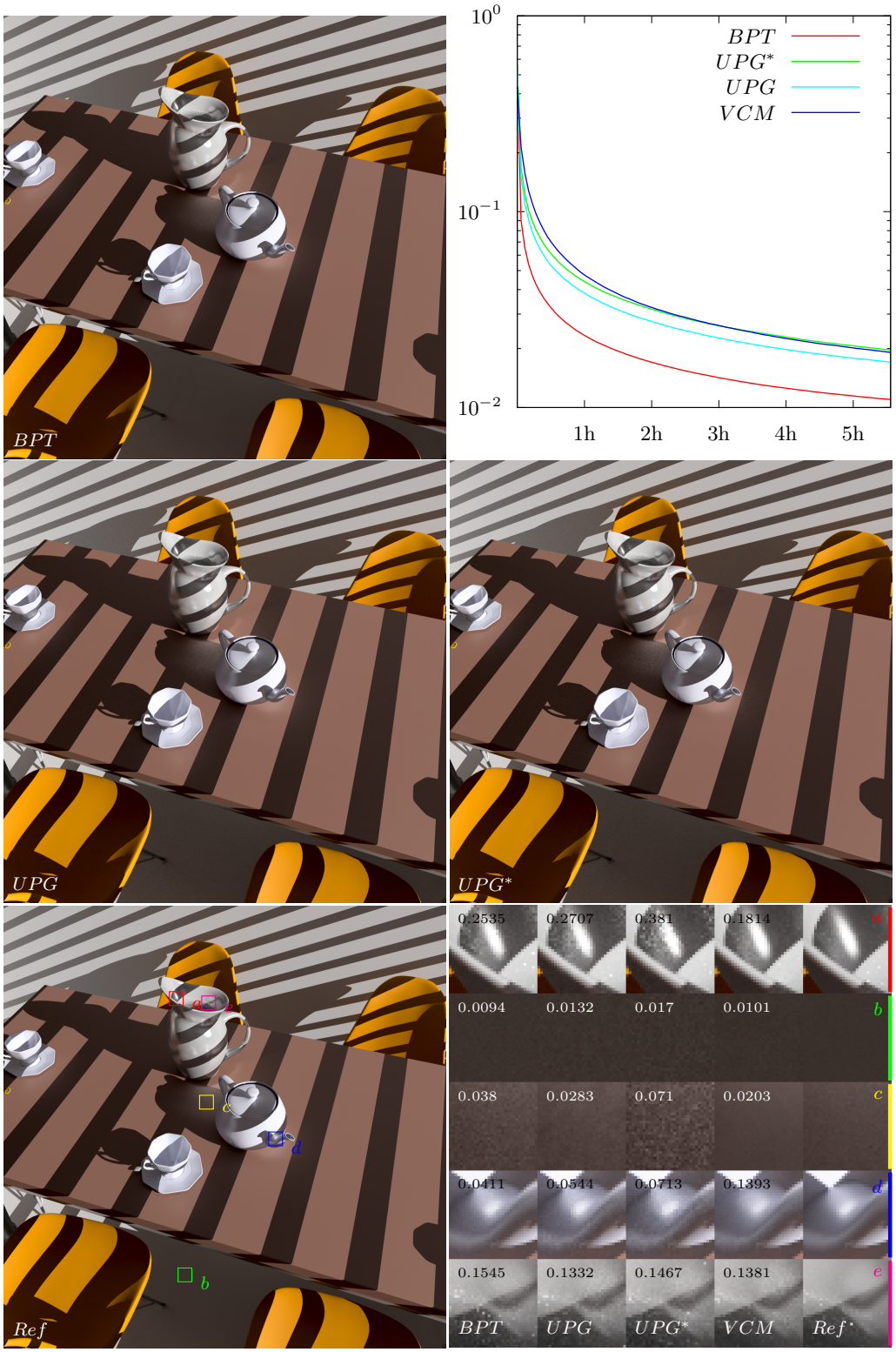

Figure 4.7: The $BreakfastRoom1^1$ scene. Another shot on the Breakfast Room scene (see Figure 4.6) from a different camera perspective. Compare the high frequency error on the table around the jar in the BPT rendering with the UPG rendering. For the explanation of the markings see Section 4.4.

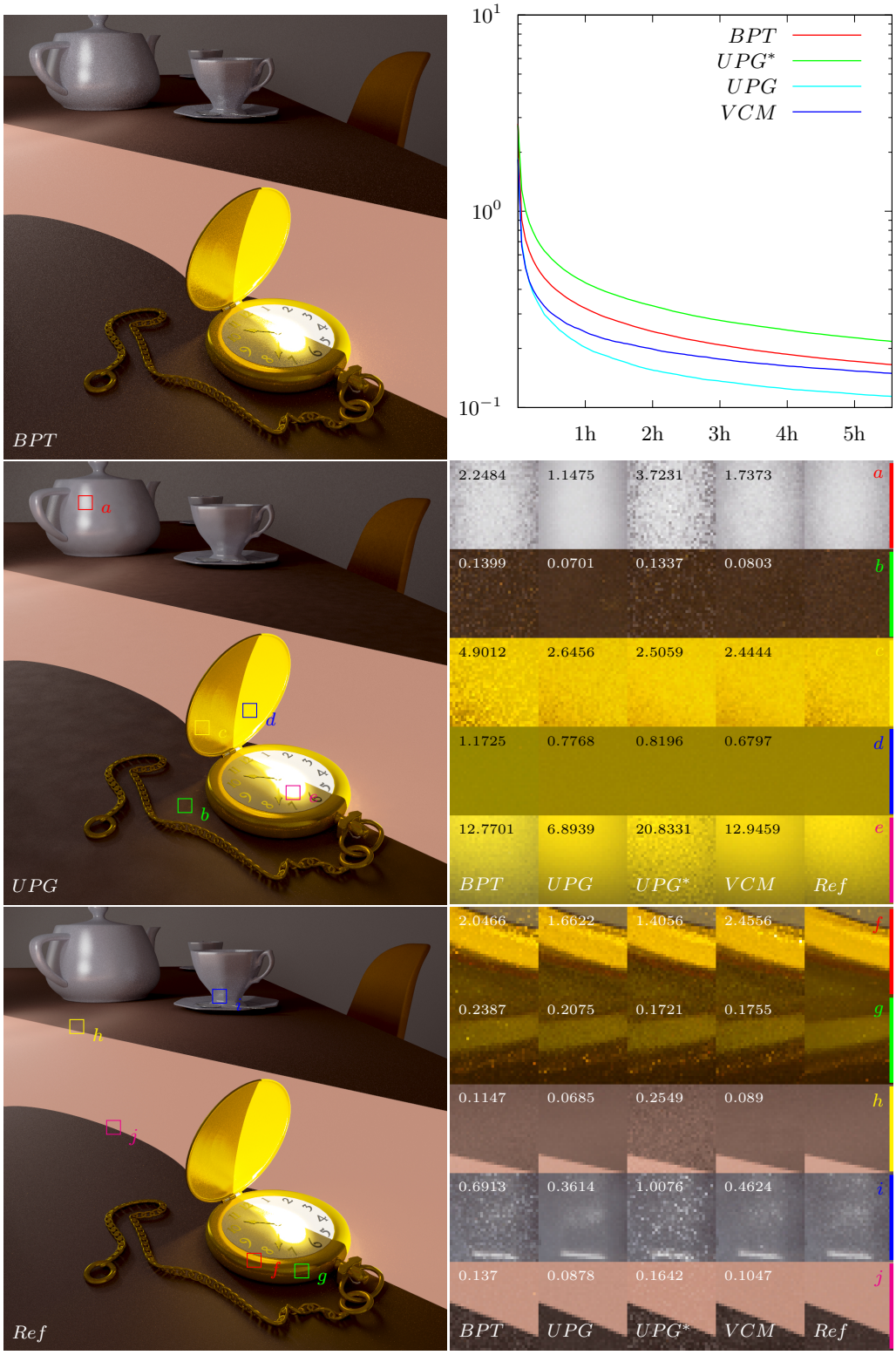

Figure 4.8: The $BreakfastRoom2^0$ scene. The scene is illuminated by directional light passing though a slightly opened doors. In low lightning conditions the result generated with UPG is actually better than the reference image generated with much more time using BPT. The excerpts *d* and *e* were tone mapped with decreased brightness to show the detail. For the explanation of the markings see Section 4.4.

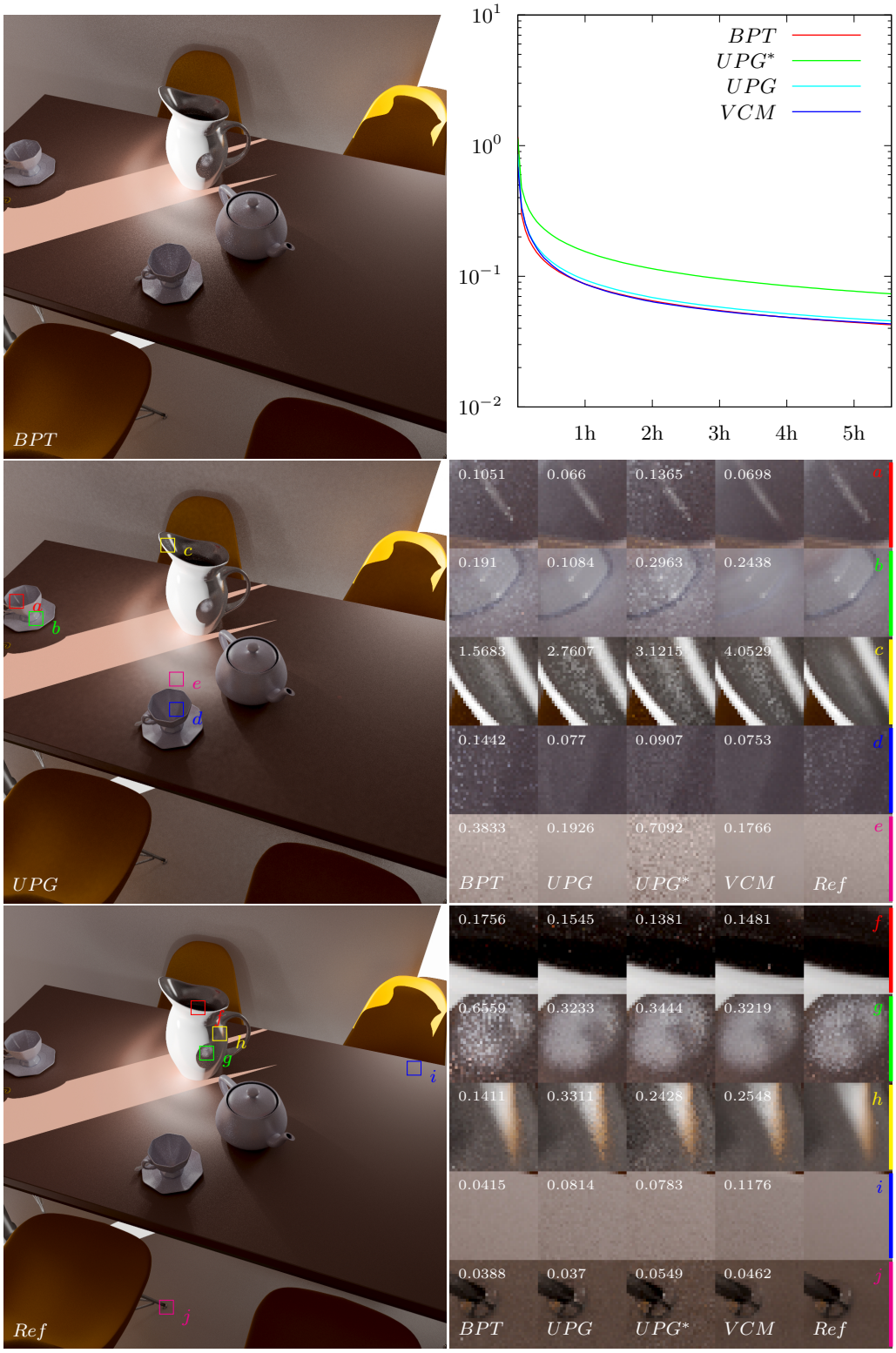

Figure 4.9: The *BreakfastRoom*2[1] scene. Another variant of the Breakfast Room scene (see Figure 4.8) from different camera perspective. The UPG rendering is significantly less noisy. Notice the characteristic "fireflies" noise on the teapot and inside the teacup in the BPT rendering. For the explanation of the markings see Section 4.4.

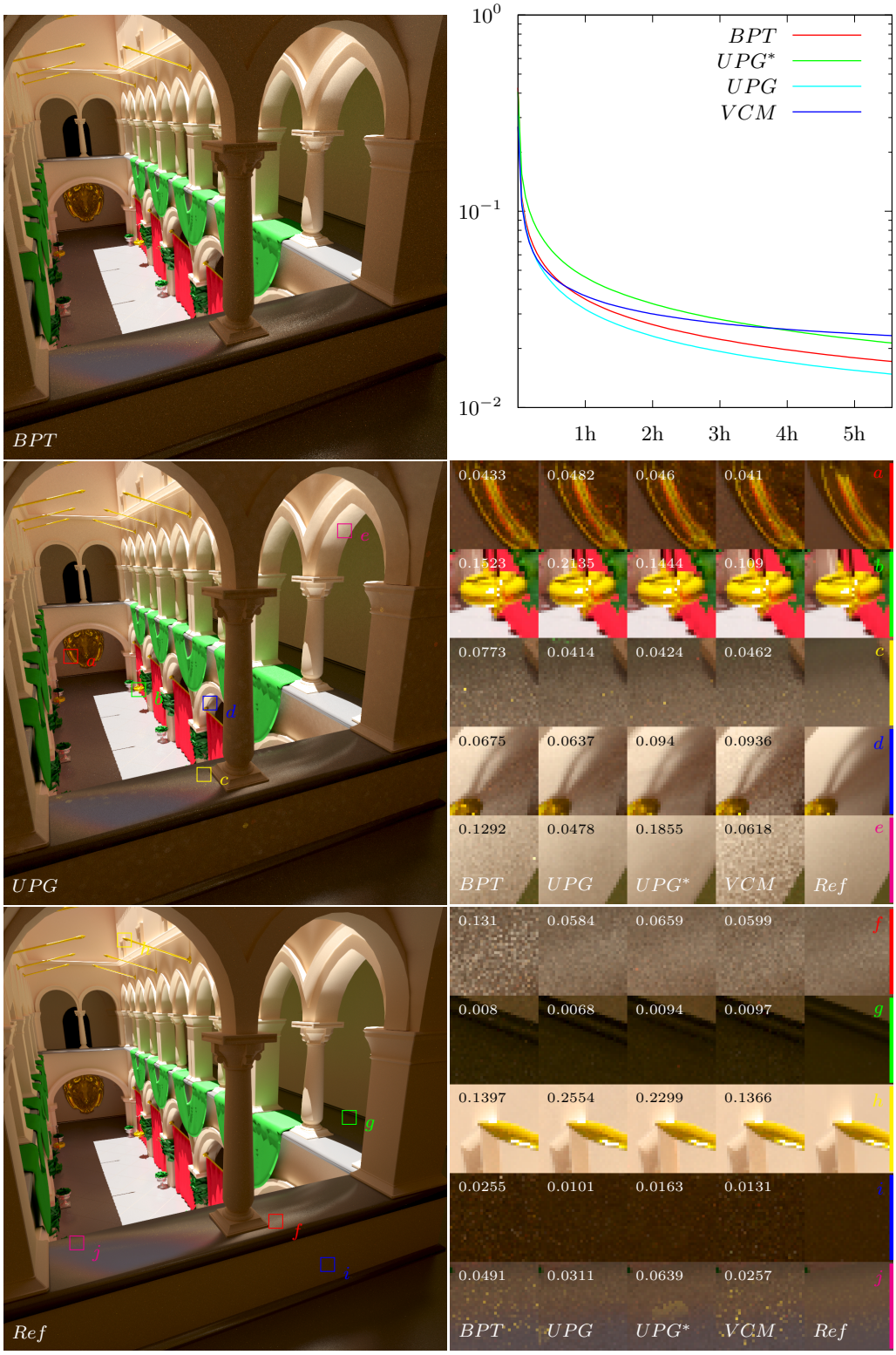

Figure 4.10: The *CrytekSponza*[0] scene. The scene is illuminated by two overhead lights. One light is a directional light simulating the sun (its direction can be guessed from the hard shadows). Another one is a diffuse area light simulating the light coming from the sky. The areas in the shadow are much less noisy in the UPG rendering than in the rest of renderings. For the explanation of the markings see Section 4.4.

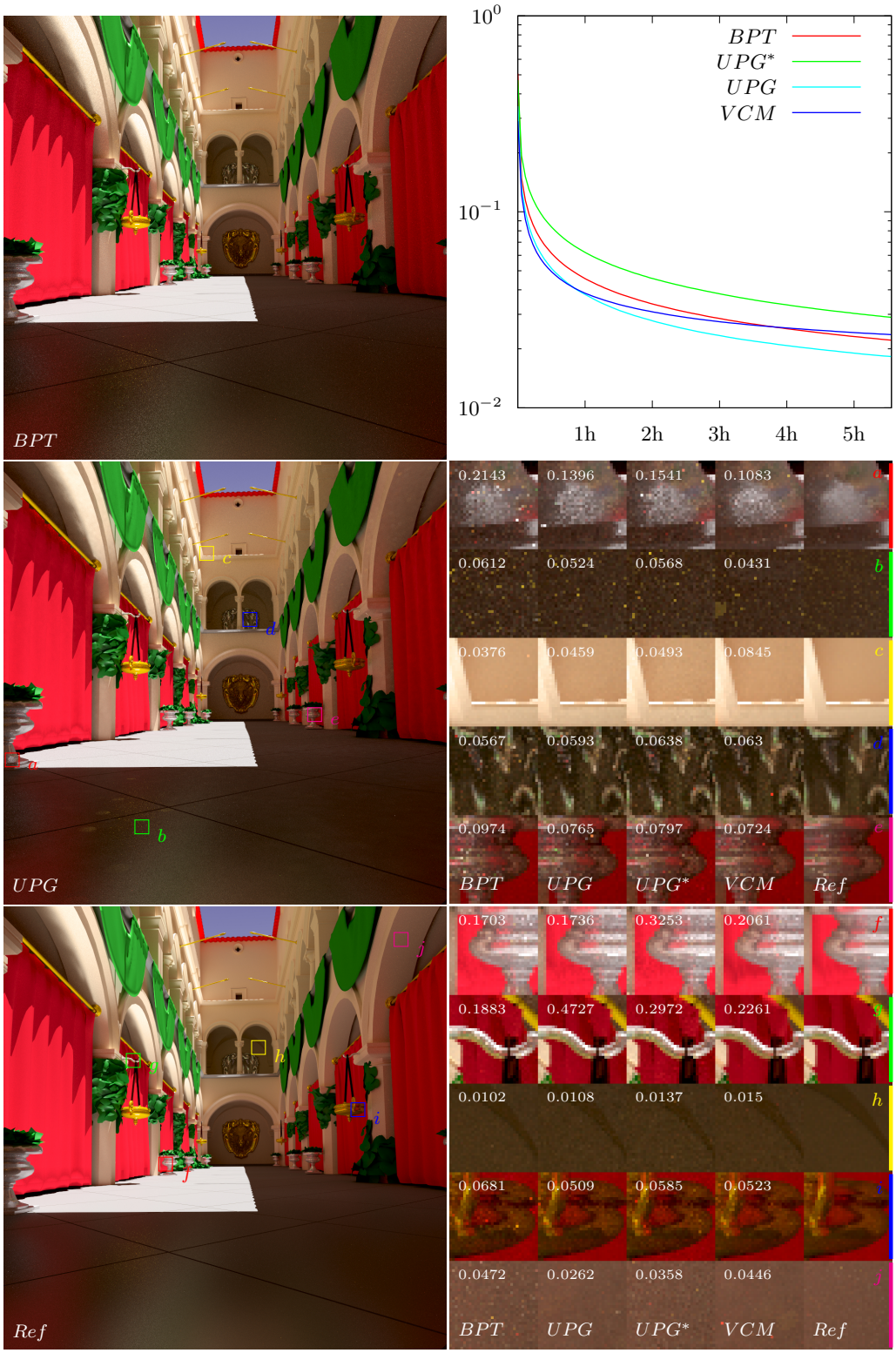

Figure 4.11: The *CrytekSponza*[1] scene. Another shot at Crytek Sponza scene. The lighting is the same as in Figure 4.10. A simple gradient sky was added to cover the black background. Notice the artifacts on the floor near the excerpt *b*. For the explanation of the markings see Section 4.4.

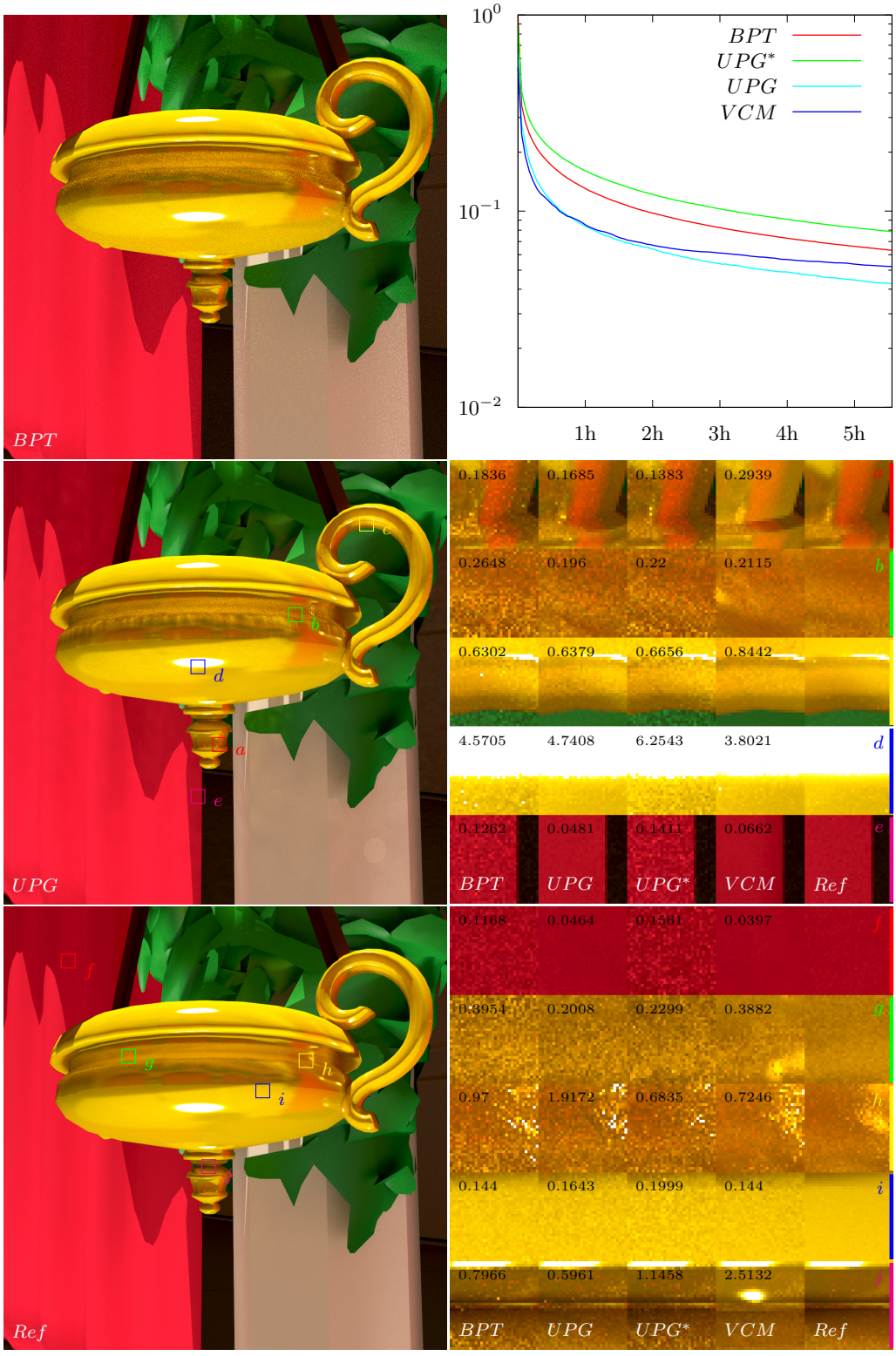

Figure 4.12: The *CrytekSponza*[2] scene. The closeup on the vase from the Crytek Sponza scene. The excerpt j shows another example of a VCM artifact. Most of the specular features are rendered more efficiently with UPG. In some cases UPG* is significantly better than UPG (excerpt h). The lighting is the same as in Figure 4.10. For the explanation of the markings see Section 4.4.

Chapter 5

Conclusions

5.1 Summary

The objective of this thesis was to explore the vertex merging based algorithms presented in the papers [QSH+15a] and [GKDS12]. Three algorithms were implemented: UPG, UPG* and VCM. Additionally BPT was implemented as a reference algorithm. The relative performance of all four algorithms was evaluated on the diverse set of scenes under various lighting conditions.

Although the method of unbiased vertex merging proposed in [QSH+15a] seems to be computationally expensive, in practice the UPG algorithm based on it offers a performance which is at least comparable and often superior to the classic approaches (represented here by BPT). Due to lack of the smoothing behavior (see Section 4.3) the UPG* isn't as efficient (for the considered scenes) as UPG. The performance of VCM is on par with the UPG but it suffers from bias artifacts. The greatest improvements of the performance for UPG were observed in the scenes with a low amount of direct light and a high amount of specular materials.

5.2 Future Work

The results presented in this thesis only scratched the surface of possibilities offered by the vertex merging techniques. It would be interesting to see the efficiency of the hybrid heuristic (combining UPG and UPG* into a single technique) proposed in [QSH+15a]. Another extension was suggested in the work [Geo12]. Georgiev describes a technique for VCM (and thus for UPG) to lower the memory consumption in cases where the BSDF structures take a significant amount of memory. Another idea is a variable gathering radius. Currently the radius is constant for the whole image. Georgiev and others in [Geo12] propose a scheme where the radius is a per-pixel property. The adjustment of super-variables like the gathering radius or multiple importance sampling beta is an area desiring additional research. An unsolved issue

is the source of the error mentioned at the end of Section 3.5.

The VCM and UPG algorithms implemented in this thesis are actually hybrids with BPT. It would be interesting to see the evaluation of the performance of the pristine versions of those techniques.

An observation could be made that the artifacts noticeable by human viewer in the VCM renderings appear only in the specific places like the corners or edges of the scene. It would be interesting to explore the idea of a heuristic combining UPG with VCM. The UPG could be only used in places where artifacts generated by VCM are visible and VCM everywhere else. The decision could be made on the length of the tentative ray. Some interpolation scheme would be needed to avoid visible transitions between the techniques.

Apart from the micro optimizations that were already applied to the implementation, there is potential for other performance improvements. The Embree library offers a functions to trace the rays in batches. An especially interesting application of the batched ray tests would be the unbiased computation of the probability density function for the vertex merging. Currently it's done a ray after ray until the hit. An introduction of extra tentative rays could improve the precision (and efficiency) of the estimation (see Section 2.4.3 for more details). The standard library functions like `pow` or `sin` in the implementation of Phong BSDF are quite expensive. The SSE versions could improve the performance of BSDF computations. Another idea for further research is implementation of the investigated techniques in the GPU.

The implementation is missing some basic algorithmic optimizations like stratification and usage of low-discrepancy sequences (Quasi Monte Carlo). Current implementation supports only basic lambertian and Phong's BSDF models. The support for more complex BSDFs and the perfectly specular surfaces for UPG are two new possible directions of the research.

The last thing is the measurement of the error in the result images itself. The absolute and root mean square errors are standard choices for basic purposes. However, it turns out that the image with a lower absolute error isn't always perceived as the better one by the human viewer. It would be interesting to explore how well the vertex merging algorithms perform when some kind of a perceptual image quality index is the measurement criterion.

Bibliography

[AK90] James Arvo and David Kirk. Particle transport and image synthesis. *SIGGRAPH Comput. Graph.*, 24(4):63–66, September 1990.

[Ant11] Dietger Van Antwerpen. Recursive mis computation for streaming bdpt on the gpu. 2011.

[BfI03] P. Bekaert and Max-Planck-Institut für Informatik. A custom designed density estimator for light transport. 2003.

[Dut03] Philip Dutré. Global illumination compendium, 2003.

[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.

[HPJ12] Toshiya Hachisuka, Jacopo Pantaleoni, and Henrik Wann Jensen. A path space extension for robust light transport simulation. *ACM Trans. Graph.*, 31(6):191:1–191:10, November 2012.

[Jen01] Henrik Wann Jensen. *Realistic Image Synthesis Using Photon Mapping.* A. K. Peters, Ltd., Natick, MA, USA, 2001.

[Kaj86] James T. Kajiya. The rendering equation. *SIGGRAPH Comput. Graph.*, 20(4):143–150, August 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.

[LW94a] Eric P. Lafortune and Yves D. Willems. A theoretical framework for physically based rendering. *Computer Graphics Forum*, 13(2):97–107, 1994.

[LW94b] Eric P. Lafortune and Yves D. Willems. Using the modified phong reflectance model for physically based rendering. 1994.

[McG17] Morgan McGuire. Computer graphics archive. `https://casual-effects.com/data`, July 2017. Accessed: 2018-02-04.

[PH10] Matt Pharr and Greg Humphreys. *Physically Based Rendering, Second Edition: From Theory To Implementation.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition, 2010.

[QSH+15a] Hao Qin, Xin Sun, Qiming Hou, Baining Guo, and Kun Zhou. Unbiased photon gathering for light transport simulation. *ACM Trans. Graph.*, 34(6):208:1–208:14, October 2015.

[QSH+15b] Hao Qin, Xin Sun, Qiming Hou, Baining Guo, and Kun Zhou. Unbiased photon gathering for light transport simulation supplementary material. 2015.

[Vea97] Eric Veach. *Robust Monte Carlo Methods for Light Transport Simulation.* PhD thesis, Stanford, CA, USA, 1997. AAI9837162.

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

[VG95] Eric Veach and Leonidas J. Guibas. Optimally combining sampling techniques for monte carlo rendering. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, pages 419–428, New York, NY, USA, 1995. ACM.

Appendix A

Help Snapshot

Description of the main functionalities can be found in Section 3.2.

```
Input file is required.

    Usage:
      master <in> [options]           Render the scene from the <in>
          ↪ file.
      master average <in>             Compute average value of pixels
          ↪ in the image <in>.
      master errors <fst> <snd>       Compute abs and rms (in this
          ↪ order) errors between the images <fst> and <snd>.
      master time <in>                Returns the rendering time of
          ↪ the image <in>.
      master measurements <in>        Extract and print measurements
          ↪ from the <in> file.
      master traces <in>              Print positions of traces
          ↪ extracted from input file metadata.
      master continue <in>            Continue rendering of the <in>
          ↪ image.
      master gnuplot <ins>...         Create convergence charts from
          ↪ multiple <ins> images.
      master diff <out> <fst> <snd>   Compute relative difference
          ↪ between <fst> and <and> and save the result as <out>.
      master merge <out> <fst> <snd>  Merge the images <fst> and <snd>
          ↪  and save the result as <out>.
      master strip <out> <in>         Strip metadata from file <in>
          ↪ and save the result as <out>.
      master bake <out> <in>          Remove the channel with number
          ↪ of samples from the image <in> save the result as <out>.

    Options (master):
      -h --help                       Show this help.
      --version                       Show version.
      --PT                            Use path tracing for rendering (
          ↪ this is default one).
      --BPT                           Use bidirectional path tracing (
          ↪ balance heuristics).
      --VCM                           Use vertex connection and
          ↪ merging.
      --UPG                           Use unbiased photon gathering.
      --num-photons=<n>               Use <n> photons. [default: 1 000
          ↪  000]
      --radius=<n>                    Use <n> as maximum gather radius
          ↪ . [default: 0.1]
```

```
  --roulette=<n>                    Russian roulette coefficient. [
      ↪ default: 0.5]
  --beta=<n>                        MIS beta. [default: 1]
  --alpha=<n>                       VCM alpha. [default: 0.75]
  --batch                           Run in batch mode (interactive
      ↪ otherwise).
  --quiet                           Do not output anything to
      ↪ console.
  --no-vc                           Disable vertex connection.
  --no-vm                           Disable vertex merging.
  --no-ui                           Disable user interface.
  --from-camera                     Merge from camera perspective.
  --from-light                      Merge from light perspective.
  --no-lights                       Do not draw the lights.
  --no-reload                       Disable auto-reload (input file
      ↪ is reloaded on modification in interactive mode).
  --num-samples=<n>                 Terminate after <n> samples.
  --num-seconds=<n>                 Terminate after <n> seconds.
  --num-minutes=<n>                 Terminate after <n> minutes.
  --parallel                        Use multi-threading.
  --output=<path>                   Output file. <input>.<width>.<
      ↪ height>.<time>.<technique>.exr if not specified.
  --reference=<path>                Reference file for comparison.
  --seed=<n>                        Seed random number generator.
  --snapshot=<n>                    Save output every <n> seconds.
  --camera=<id>                     Use camera with given id. [
      ↪ default: 0]
  --resolution=<WxH>                Resolution of output image. [
      ↪ default: 512x512]
  --trace=<XxY[xW]>                 Trace errors in window of radius
      ↪  W and at the center at XxY. [default: XxYx2]
  --sky-horizon=<RxGxB>             Color of sky horizon. [default:
      ↪ 0x0x0]
  --sky-zenith=<RxGxB>              Color of sky zenith. [default: 0
      ↪ x0x0]
  --blue-sky=<B>                    Alias to --sky-horizon=<0x0x0>
      ↪ --sky-zenith<0x0xB>. [default: 0]

Options (gnuplot):
  --input=<path>                    The path to .exr file containing
      ↪  error data.
  --output                          The output image with the chart.
  --traces                          Generate a matrix of charts with
      ↪  data from window traces.
  --select=<wildcard>               Consider only series which name
      ↪ matches <wildcard>.

Examples:
  View the image.exr file.
    master image.exr

  Render scene.blend using UPG with vertex merging from camera
      ↪ perspective.
    master scene.blend --UPG --radius=0.01 --beta=2 --from-camera
        ↪ --output=image.exr

  Render scene.blend using BPT for 20 minutes, save the result
      ↪ every 6 minutes.
    master scene.blend --BPT --output=image.exr --num-minutes=20
        ↪ --snapshot=360

  Render scene.blend using BPT use all cores, use camera no. 2.
    master scene.blend --BPT --output=image.exr --parallel --
        ↪ camera=2
```

```
Render scene.blend using UPG use reference.exr as the reference
    ↪ image, compute errors in square window at position [32,
    ↪ 32] with side size equal to 16.
  master scene.blend --UPG --output=image.exr --reference=
      ↪ reference.exr --trace=32x32x8

Render scene.blend using UPG use all cores compute errors for
    ↪ multiple windows.
  master scene.blend --UPG --output=image.exr --reference=
      ↪ reference.exr --trace=32x32x8 --trace=42x142x16

Merge two results from two different images.
  master output.exr image-machineA.exr image-machineB.exr

Render image in batch (headless) mode.
  master scene.blend --BPT --output=image.exr --parallel --
      ↪ camera=2 --batch

Render simple gradient based sky in places where there is no
    ↪ geometry.
  master scene.blend --BPT --output=image.exr --parallel --batch
      ↪  --blue-sky=10
```

Appendix B

Optimal Radii

Alg.	Scene	0.005	0.01	0.015	0.02	0.025	0.03	0.035	0.04	0.045	0.05	0.055	0.06	Exp.
UPG	$Bathroom^0$	1.08	1.09	1.10	1.11	1.10	1.06	1.05	1.02	1.02	1.00	1.00	1.00	$\times 10^0$
UPG	$Bearings^0$	5.84	4.67	4.26	4.09	4.06	4.00	4.14	4.33	4.21	-	4.41	4.70	$\times 10^{-3}$
UPG	$BreakfastRoom1^0$	6.10	5.62	5.75	6.05	6.74	7.40	8.25	9.18	9.95	-	-	-	$\times 10^{-2}$
UPG	$BreakfastRoom1^1$	6.45	5.81	5.83	6.07	6.72	7.45	8.13	9.04	9.88	-	-	-	$\times 10^{-2}$
UPG	$BreakfastRoom2^0$	3.89	2.95	2.85	2.91	3.12	3.39	3.61	3.94	4.16	-	-	-	$\times 10^{-1}$
UPG	$BreakfastRoom2^1$	1.94	1.51	1.38	1.33	1.35	1.40	1.48	1.56	1.67	-	-	-	$\times 10^{-1}$
UPG	$CrytekSponza^0$	7.23	5.35	4.76	4.57	4.59	4.73	5.01	5.26	5.59	-	-	-	$\times 10^{-2}$
UPG	$CrytekSponza^1$	8.49	6.24	5.64	5.53	5.57	5.78	6.18	6.35	6.99	-	-	-	$\times 10^{-2}$
UPG	$CrytekSponza^2$	1.74	1.27	1.17	1.16	1.20	1.26	1.35	1.44	1.54	-	-	-	$\times 10^{-1}$
UPG^*	$Bathroom^0$	1.07	1.09	1.10	1.10	1.11	1.08	1.08	1.07	1.07	1.08	1.10	1.12	$\times 10^0$
UPG^*	$Bearings^0$	6.22	5.71	5.33	5.24	5.24	5.32	5.57	5.60	5.95	-	6.19	6.51	$\times 10^{-3}$
UPG^*	$BreakfastRoom1^0$	6.44	6.27	6.81	7.48	8.59	9.52	10.42	11.58	12.70	-	-	-	$\times 10^{-2}$
UPG^*	$BreakfastRoom1^1$	6.71	6.68	7.26	8.13	9.17	10.34	11.64	12.83	13.96	-	-	-	$\times 10^{-2}$
UPG^*	$BreakfastRoom2^0$	6.70	6.10	6.29	6.71	7.21	7.84	8.35	8.82	9.28	-	-	-	$\times 10^{-1}$
UPG^*	$BreakfastRoom2^1$	2.26	2.25	2.47	2.76	3.08	3.37	3.71	4.00	4.28	-	-	-	$\times 10^{-1}$
UPG^*	$CrytekSponza^0$	7.68	6.93	7.60	8.60	9.43	10.46	11.31	12.33	13.06	-	-	-	$\times 10^{-2}$
UPG^*	$CrytekSponza^1$	9.95	9.22	10.41	11.91	13.11	14.66	16.02	16.98	18.24	-	-	-	$\times 10^{-2}$
UPG^*	$CrytekSponza^2$	2.49	2.20	2.30	2.45	2.62	2.82	3.04	3.18	3.39	-	-	-	$\times 10^{-1}$
VCM	$Bathroom^0$	-	-	-	-	-	-	-	-	-	-	-	-	$\times 10^0$
VCM	$Bearings^0$	-	8.89	8.16	7.49	7.69	7.90	6.73	6.68	6.85	6.84	-	-	$\times 10^{-3}$
VCM	$BreakfastRoom1^0$	-	7.16	6.80	6.50	6.46	6.62	6.77	7.10	7.56	7.95	-	-	$\times 10^{-2}$
VCM	$BreakfastRoom1^1$	-	1.03	1.03	1.02	1.02	1.02	1.02	1.02	1.02	1.02	1.02	1.02	$\times 10^0$
VCM	$BreakfastRoom2^0$	-	4.02	3.79	3.94	4.48	4.99	5.47	6.17	6.68	7.19	-	-	$\times 10^{-1}$
VCM	$BreakfastRoom2^1$	-	3.30	3.31	3.32	3.30	3.29	3.29	3.31	3.31	3.29	3.29	3.33	$\times 10^0$
VCM	$CrytekSponza^0$	-	7.86	6.58	5.99	5.71	5.61	5.60	5.77	5.97	6.22	-	-	$\times 10^{-2}$
VCM	$CrytekSponza^1$	-	4.37	4.32	4.31	4.31	4.31	4.30	4.31	4.32	4.32	-	-	$\times 10^{-1}$
VCM	$CrytekSponza^2$	-	8.02	7.99	7.99	7.98	7.98	7.98	7.98	7.98	7.99	-	-	$\times 10^{-1}$

Alg.	Scene	0.06	0.065	0.07	0.08	0.09	0.1	0.11	0.12	0.13	0.14	0.2	0.3	Exp.
VCM	$Bathroom^0$	-	-	6.4712	6.2093	6.0795	6.0339	6.1291	6.2458	6.4321	6.6574	8.4883	12.4645	$\times 10^{-1}$

Table B.1: The absolute error with respect to the reference image for different radii (see Section 4.2 for details). The value highlighted with the green color is the minimum error. The errors from the **different** rows cannot be compared as they may have been rendered on different machines or with different rendering time.