MAX-PLANCK-GESELLSCHAFT

Max-Planck-Institut für Informatik
Computer Graphics Group
Stuhlsatzenhausweg 85
66123 Saarbrücken, Germany

Advanced Density Estimation Techniques for Global Illumination

Master Thesis in Computer Science

Computer Science Departement
University of Saarland

Submitted by **Robert Herzog, B.Sc.**

Conducted under Advice of
Dr. Vlastimil Havran
and
Prof. Dr. Hans-Peter Seidel

Begin: April 20, 2005
End: October 20, 2005

Saarbrücken, 6th December 2005

Statement under Oath

I, Robert Herzog (student of Computer Science at the University of Saarland, student ID no. 2503674), solemnly declare that I have written this thesis independently, and that I did not make use of any aids other than those acknowledged in this thesis. Neither this thesis, nor any similar work, have previously been submitted to any examination board.

Robert Herzog

Contents

List of Figures

List of Tables

Nomenclature

Functions and Scalars

x	Location and query point
$f(x)$	Density function
$\hat{f}(x)$	Density estimator
$\hat{f}(x, h)$	Density estimator with bandwidth h
$\mathcal{K}$	Kernel function
$\mathcal{K}_h$	Scaled kernel function
h	Bandwidth, window width, or support
n	Number of samples or observations
X_i	Sample of random variable X
$d_k(x)$	Maximum distance between x and the k nearest neighbors
θ	Azimuth angle
ϕ	Longitude angle
R	Gather radius or bandwidth in 2D
f_r	Bidirectional Reflectance Distribution Function (BRDF)
ρ_d	Material's diffuse reflectivity ($\rho_d \in [0..1]$)
ρ_s	Material's specular reflectivity ($\rho_s \in [0..1]$)

Sets and Vectors

$\vec{n}$	Normal vector
$\vec{\omega_i}$	Incoming ray direction (θ_i, ϕ_i)
$\vec{\omega_o}$	Reflected (outgoing) ray direction (θ_o, ϕ_o)
L	Radiance
L_d	Diffuse component of the radiance
Φ	Flux
$A[i]$	i-th element of Array A
Ω	Set of directions over the hemisphere
A	Area of points

Operators and Other Symbols

kNN	k nearest neighbors
KDE	Kernel density estimation
MSE	Mean square error
ISE	Integrated square error
$MISE$	Mean integrated square error
$AMISE$	Asymptotic mean square error
E	Expected value
Var	Variance
PDF	Probability density function
$o(b_n)$	Sequence $a_n = o(b_n)$ as $n \to \infty,$ if $\limsup_{n\to\infty} \lvert a_n/b_n \rvert < \infty$
PM	Photon mapping
NPM	Normal photon mapping *à* la Henrik Wann Jensen
RPM	Reverse photon mapping
FGR	Final gather ray

Acknowledgment

First of all I want to thank Vlastimil Havran for his steady support and for reading this thesis. Besides helping me with constructive suggestions and critics, he also provided me many useful books and papers for my thesis. A special thanks to Jiří Bittner who was involved in the ray map project. Further, I would like to thank Prof. Hans-Peter Seidel for giving me the opportunity to work at the MPII in the Computer Graphics Group. I want to thank Velislava for revising this thesis, my sister Maria who helped me with the layout, and my parents for supporting my studies. And last but not least, I should not forget the funny breaks I had with my flat-mate Philipp during writing this thesis at home.

Chapter 1

Introduction

Computer graphics has always been a large field in computer science and is steadily advancing from the research niche to commercial products in the entertainment industry, medicine, production and simulation, advertisement, robotics. In contrast to other computer science fields, computer graphics (CG) can be very appealing and challenging because of its beautiful and satisfactory results. It comprises a large variety of knowledge from computer science, mathematics, and physics. The probably largest clientele of CG is still the entertainment industry consisting of movie and computer games companies. However also other branches have become aware of the utility of computer graphics, for example the quality control and automatization of production or the technical assistance in the medicine. New movies often make extensive use of computer generated special effects or are even virtually rendered entirely from 3D computer models. The entry of computer rendering to industry also opened a new era for arts: nowadays artists often do not work with brushes and paint but prefer the use of computer tablets in combination with artistic software to simulate all kind of painting styles and instruments in a clean and cheap manner. And the results are often indistinguishable from real drawings. Besides the classic 2D drawing, the virtual 3D modeling and rendering is becoming an emerging style in arts. In this thesis, we will not be concerned with 3D modeling or acquisition but only with the realistic visualization of such, including all kinds of lighting effects. A 3D computer graphics model alone, consisting of vertices and (textured) faces, is usually visually unpleasing. This is because the human eye, which is used to see photo realism in a 3D environment, inherently detects the illumination error. Therefore, the ”final product” is often presented using photo realistic rendering. Due to the explosion of graphics hardware performance in the last few years, interactivity plays an important role in the 3D visualization. Computer games are presenting increasingly realistic worlds in real-time and it is only a matter of time till realistic global illumination effects are considered in games. Nevertheless, photorealism does not come for free and is one of the toughest problems in computer science. Since the human eye is the most significant and developed sense, it is difficult to mislead it and one has to simulate global illumination which is very complex in realistic settings. Global illumination can be understood as a gigantic parallel machine tracing a huge amount of photons simultaneously. This is impossible to simulate by nowadays computers. Therefore, clever algorithms were developed to approximate reality efficiently, which we will briefly discuss in the next chapter. This thesis is dedicated to one of these methods called photon mapping and includes *two* improvements of the concept in more detail with the aim to give the reader some new insights.

1.1 Overview

The thesis is divided into four main topics:

1. background information about global illumination and density estimation,
2. photon mapping and photon map data structures,
3. reverse photon mapping,
4. and the ray map.

The thesis is organized in a way an uninitiated reader can understand and appreciate the two new methods, *reverse photon mapping* and the *ray map*. The advanced reader may skip the next two chapters, which are intended to provide the background information of global illumination and density estimation. After a review of the physics of light and the basics of global illumination in Chapter 2, we continue with a rather theoretical introduction to statistical density estimation in Chapter 3. Chapter 4 describes the fundamentals of photon mapping as a specific application of density estimation in computer graphics. Besides the classical photon mapping algorithm, several novel enhancements and ideas are discussed. Chapter 5 gives an overview and a comparison of a variety of search data structures applicable to photon mapping and provides a detailed description including pseudo code for our new proposed kd-tree structure. Chapter 6 presents a novel application of photon mapping combined with Monte Carlo ray tracing which we designed to speed up the rendering of indirect illumination. This method named *reverse photon mapping* has been published to Eurographics in August 2005 [29]. Chapter 7 shows quite a different approach to photon density estimation intended to overcome certain bias sources inherent to photon mapping. This method has been published to Eurographics Symposium on Rendering in June 2005 [28].

Chapter 2

Introduction to Global Illumination

Photorealistic rendering has been one of the driving forces in computer graphics over the last decades. The probably most technical part of realistic image synthesis is *global illumination*. The word ”global” refers to the fact that the illumination of an object depends on the light it receives from all other (non-occluded) objects. In this sense, computing the illumination even at a single point requires consideration of the entire scene model. Despite of this complexity of global illumination, it is evolving quickly. Specially designed global illumination methods and hybrid approaches make use of programmable graphics hardware, which allows global illumination, once solely used for scientific simulation of single images, to open up for use in animations. Nevertheless, including global illumination in complex scenarios is generally difficult and the practical usage in industry is still moderate. Photon mapping in particular temporal photon mapping is one method that opened the way for practical global illumination in animations. But also radiosity related techniques especially instant radiosity are becoming popular again due to the modern programmable graphics hardware. In recent years there have been many attempts of real-time (partial) global illumination solutions which all have pros and cons. The trend moves towards fast and interactive (hybrid) algorithms that are suitable for rasterization on graphics hardware neglecting accuracy and bias in the results. We will not regard rasterization methods although they have become more popular even in high-quality rendering due to the rapidly increasing processing power and accuracy of commodity graphics accelerators. In this thesis we will focus on the accurate and more general methods used for production rendering in the industry. The theory induced by the nature of light stays the same for all algorithms and cannot be tricked by simple heuristics aiming at interactivity. Therefore, it is necessary to understand the nature of light in order to simulate global illumination correctly. First, we recall the physical basics of light and global illumination before we proceed with specific algorithms.

2.1 The Nature of Light

Light is a complicated phenomena and it is still not completely understood. Since the earliest days human beings have been fascinated by the perception of light because it is our most powerful sense. Many theories explaining *light* have been proposed over the centuries. In former days the Greeks believed that ”light” starts in our eye which is emanating rays that touch the environment we see. Surprisingly, this idea has relived and is exploited in Computer Graphics as we will see. It took about 1000 years before the scientist Alhazen (A.D. 956-1038) described the light by straight rays and modeled the eye with a pinhole camera called *camera obscura.* More than 600 years later, Christiaan Huygens and Isaac Newton came up with two different novel theories. Huygens demonstrated the wave optics of light whereas the more prominent Newton supported his light particle theory and argued heavily against Huygens wave model. From the 19th century on, researchers like Augustin Fresnel and Thomas Young studied effects as polarization and diffraction. James Maxwell further described the properties of electromagnetic waves and later on Albert Einstein introduced the photon and the photo-electric effect. At that time it became clear that light could not be explained by either theory. Niels Bohr formalized the *dualism of light* established by the field of quantum mechanics. Therein, light is classified into ray/wave optics, and photon (particle) optics. In computer graphics we are mostly concerned with a highly abstract model based on ray optics even for photon mapping. Furthermore, we neglect the speed of light. Nevertheless, most light phenomena that we are able to see in our daily life can be simulated in a realistic way by computer graphics.

2.1.1 Light Terminology

In order to describe light and the transport of energy, a basic terminology must be introduced. There are two different terminologies, the *radiometry* which is physically based and the *photometry*, which was derived from the human perception of light. We will first introduce the radiometry which is more accepted in science and generally used in computer graphics. The smallest quantity in lighting is the photon. The energy of a photon with wavelength λ is $e_\lambda = \frac{h \cdot c}{\lambda}$ where h is the Planck's constant $h \approx 6.63 \cdot 10^{-34} J \cdot s$ and c is the speed of light.

2.1.1.1 Radiant Energy

Radiant energy Q is the total energy gathered from all photons of all wavelengths λ

$$Q = \int_0^\infty n_\lambda e_\lambda d\lambda \, [W \cdot s] \,. \tag{2.1}$$

2.1.1.2 Radiant Flux

Radiant flux Φ is defined as radiant energy per time

$$\Phi = \frac{dQ}{dt} \, [W] \tag{2.2}$$

It represents the time rate of flow of radiant energy.

2.1.1.3 Irradiance/ Radiosity

The general terminology *radiant flux area density* is usually separated into *irradiance* E and *radiosity* B, also known as radiant exitance. It represent the flux density on a surface which is defined as radiant flux per differential area

$$E(x) = \frac{d\Phi}{dA} \left[\frac{W}{m^2} \right] . \tag{2.3}$$

The difference between irradiance and radiosity is that irradiance corresponds to the flux arriving on a surface and radiosity corresponds to the flux leaving the surface.

2.1.1.4 Radiant Intensity

Radiant intensity I represents the directional density of flux (i.e. the flux coming from a certain direction) and is defined as flux per differential solid angle $d\vec{\omega}$

$$I(d\vec{\omega}) = \frac{d\Phi}{d\vec{\omega}} \left[\frac{W}{sr}\right] \tag{2.4}$$

The solid angle is explained in Section 2.1.2.

2.1.1.5 Radiance

Radiance L can be considered as a product of directional density and area density of flux which is dependent on the surface orientation (therefore the division by the cosine of the azimuth angle θ). It is probably the most important quantity in computer graphics because this is what we measure with ray tracing. It is defined as radiant flux per differential solid angle per differential projected area.

$$L(x,\vec{\omega}) = \frac{d^2\Phi}{\cos\theta dA d\vec{\omega}} \left[\frac{W}{m^2 \cdot sr}\right] \tag{2.5}$$

Informally, it can be understood as the number of photons per time arriving from a certain direction $\vec{\omega}$ through the solid angle $d\vec{\omega}$ in a small area dA. Therefore, radiance is a five-dimensional quantity. Another important aspect is that radiance stays constant along a line in vacuum. We can also refine the radiance definition to include the wavelength, which gives us the *spectral radiance* L_λ, the radiance for a certain differential wavelength.

2.1.1.6 The Relationship between Radiometric Quantities

From the definitions of the radiometric quantities, the following relationship can be derived:

$$\Phi = \int_A E(x)dx = \int_A \int_\Omega L(x,\vec{\omega})(\vec{\omega}\cdot\vec{n})d\vec{\omega}dx, \tag{2.6}$$

where $\vec{n}$ is the surface normal at x and $\vec{\omega}\cdot\vec{n} = \cos\theta$. If the complete radiance field on a surface is available then the irradiance/radiosity can be computed by integrating the incident/exitant radiance field over all directions. The incident/exitant flux is computed by integrating the irradiance/radiosity over the area.

2.1.1.7 Photometry

The difference between radiometry and photometry is that photometric quantities include the visual response of the average observer. For example, luminous flux Φ_v is the visual response to radiant flux

$$\Phi_v = \int_\Lambda \Phi_\lambda V(\lambda)d\lambda, \tag{2.7}$$

where $V(\lambda)$ is the visual response of a standard observer and Λ is the visible spectrum ($\approx$ 380 nm – 780 nm). *Illuminance*, E_v, is the counterpart to irradiance and *luminous exitance*, M_v, to radiosity or radiant exitance. *Luminous intensity*, I_v, is the photometric quantity that corresponds to radiant flux and *luminance* is the photometric equivalent of radiance. The radiometric quantities are often preferred in global illumination programs but the visual response plays an important role for post-processing rendered images via *tone mapping*.

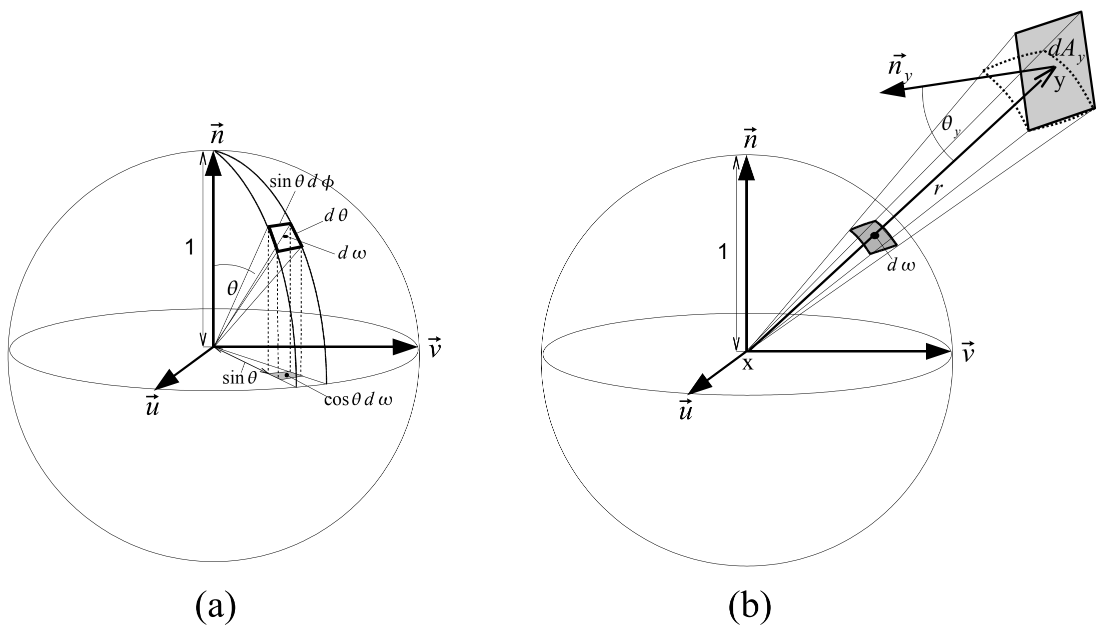


Figure 2.1: *(a) The area subtended by a differential solid angle $d\vec{\omega}_\theta$ is the product of the differential length of the longitude arc $d\theta$ and the latitude arc $\sin\theta d\phi$. (b) The differential solid angle $d\vec{\omega}$ subtended by a differential area dA_y is equal to $dA_y \cos\theta_y / r^2$, where θ_y is the angle between dA_y's surface normal and the vector to the point x and r is the distance from x to y. In other words, $d\vec{\omega}$ is the projection of the differential area dA_y onto the unit sphere around point x.*

2.1.2 Solid Angle

In order to integrate functions over the hemisphere which becomes necessary for computing radiometric quantities, a specific measure defined on the hemisphere is needed. This is the *solid angle*. The finite solid angle Ω corresponds to the projection of an object onto the unit (hemi-)sphere:

$$\Omega = \frac{A}{r^2}, \tag{2.8}$$

where A is the projected area on the sphere with radius r. The solid angle is dimensionless but is expressed in *steradians* [sr]. The differential solid angle can be thought of as a combination of the direction of a beam and its angular size defined over the unit hemisphere. The direction is expressed in spherical coordinates (θ, ϕ), corresponding to the azimuth and longitudinal angle. The size of a differential solid angle is given by $d\vec{\omega} = \sin\theta d\theta d\phi$, which is defined as the infinitesimal area on the unit sphere computed as the product of the longitude arc $d\theta$ and the latitude arc $\sin\theta d\phi$ (see Figure 2.1 (a)). The term $\sin\theta$ comes from the parameterization of the sphere to spherical coordinates because the differential area $d\theta \cdot d\phi$ on the hemisphere decreases with respect to the sine of the angle θ when it moves towards the pole. The transformation from a differential surface with differential area dA_y, distance r_y and orientation θ_y to a differential solid angle $d\vec{\omega}_\Theta$ is given by

$$d\vec{\omega}_\Theta = \frac{\cos\theta_y dA_y}{r_y^2}. \tag{2.9}$$

This relationship is visualized in Figure 2.1 (b).

2.2 Light-Matter Interaction

Light alone is not very valuable for us, but the interaction with the scene model (the illumination) shows the variety of our environment. Looking at arbitrary real surfaces reveals the complexity of such light interaction. In real world, light is reflected and transmitted in numerous ways depending on its wavelength, the incident and reflected direction. This usually also varies across the surface and depends on the incoming radiance point and even the reflection location. The general light-surface interaction becomes a 10-dimensional problem. To describe this kind of scattering by a mathematical function suitable for simulation with a computer is generally difficult because of the high-dimensionality of the problem. Therefore, one reduces the dimension from 10 to 8 dimension neglecting wavelength and time. Functions of this general type are still inappropriate for efficient representation and simulation by a computer. Besides many materials especially artificial ones do not change across the surface and very few materials exhibit sub-surface scattering effects. Therefore, depending on the material context the general scattering function is further reduced to 6 dimensions discarding either spatial variation or assuming that incident and exitant location of the light scattering event are the same (i.e. no sub-surface scattering). The former type of functions is called *bidirectional texture function* (BTF) and the latter type is classified as *bidirectional scattering surface reflectance distribution function* or short BSSRDF. So far little research has been done in direction of BTFs ([15, 56]) since the acquisition of such 6D function is an expensive and time consuming process. The measured data can become very large (several GBytes) for a single material but is often quite redundant, which imposes various compression schemes.

The BSSRDF accounts for almost all scattering effects of light such as sub-surface scattering in translucent materials (e.g. milk, skin, snow, alabaster) excluding time- (e.g. phosphorescence) and wavelength-dependent effects (e.g. fluorescence). Nevertheless, it is hardly used in practice since most materials are either opaque or transparent. BSSRDFs are still too complex for efficient simulation and only addressed by a few papers.

In many cases the materials are invariant with respect to sub-surface scattering and the two spatial degrees of freedom in the BTF can be solved by a surface subdivision to piecewise constant materials. Another approach is to approximate the two spatial variation in the general scattering function by the use of 2D textures approximating various spatial material properties such as surface albedo, reflectance, transparency. In this case the scattering function is restricted to 4 dimensions and a new name is assigned for this representation: *bidirectional scattering distribution function* (BSDF) or simply BDF. In lighting simulation it is a common practice to separate reflecting and transmitting properties of a material. This is why BSDFs of reflected and transmitted light are usually considered separately and are called BRDF *Bi-directional Reflectance Distribution Function* and BTDF *Bi-directional Transmittance Distribution Function* respectively. However, we will not regard transmitting properties of a material.

2.2.1 The BRDF

The BRDF, denoted by f_r, describes the interaction of light with a surface neglecting sub-surface scattering events. We simply assume the light is reflected at the same location it is arriving. This is not always true and real objects such as human skin, milk, or alabaster reflect light in

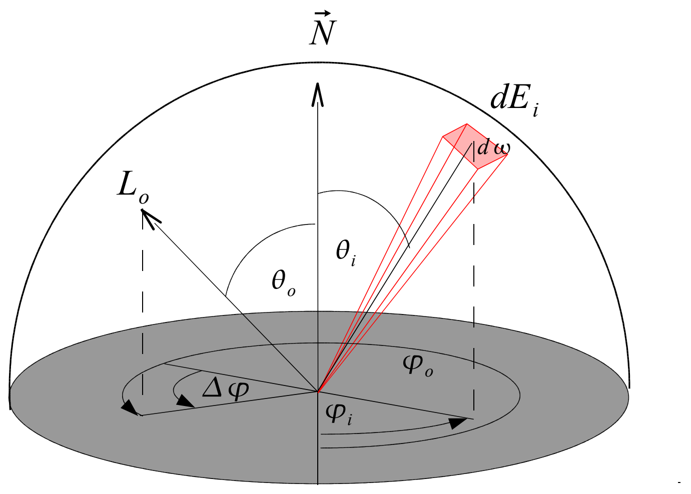


Figure 2.2: *A geometric interpretation of the BRDF for a point on a surface. The BRDF function gives the ratio of radiance L_o reflected in direction (θ_o, ϕ_o) to the differential irradiance E_i from differential solid angle $d\vec{\omega}$ of direction (θ_i, ϕ_i). In case of an isotropic BRDF, only the difference $\Delta\phi = \phi_o - \phi_i$ in the longitudinal angles ϕ_i and ϕ_o is regarded.*

the neighborhood of the incident illuminated point. The BRDF is defined as

$$f_r(x, \vec{\omega}_i, \vec{\omega}_o) = \frac{dL(x, \vec{\omega}_o)}{dE(x, \vec{\omega}_i)} = \frac{dL_o(x, \vec{\omega}_o)}{L_i(x, \vec{\omega}_i) \cos\theta_i d\vec{\omega}_i} \tag{2.10}$$

and tells us what fraction of the incoming irradiance E_i from direction $\vec{\omega}_i$ over the infinitesimal small solid angle $d\vec{\omega}_i$ is reflected at surface point x towards $\vec{\omega}_o$. The BRDF can be regarded as a probability density function which describes the probability of a photon coming from direction $\vec{\omega}_i$ to be reflected in direction $\vec{\omega}_o$. However, the difference is that it does not need to integrate to one over the hemispherical directions but can rather yield any value between zero (= total absorption) and one (= no absorption). Another important issue is that the BRDF can take any positive value even approaching infinity in case of a perfect (however unnatural) mirror where all but *one infinitesimal small* direction are zero resulting in a *dirac impulse*. Hence, having the full (hemispherical) incident radiance field available, we can compute the local illumination model, the reflected radiance for all directions $\vec{\omega}_o$:

$$L_r(x, \vec{\omega}_o) = \int_\Omega f_r(x, \vec{\omega}_i, \vec{\omega}_o) dE(x, \vec{\omega}_i) = \int_\Omega f_r(x, \vec{\omega}_i, \vec{\omega}_o) L_i(x, \vec{\omega}_i) \cos\theta_i d\vec{\omega}_i, \tag{2.11}$$

where the cosine term is computed by a dot product of the normal at surface point x and the incoming direction θ_i: $\cos\theta_i = \vec{\omega}_i \cdot \vec{n}_x$.

The BRDF has four dimensions and depends on four angles $(\theta_i, \phi_i, \theta_o, \phi_o)$ giving 4 degrees of freedom. However, for most materials the rotation around the surface normal is often superfluous. In such case of isotropic materials only the relative longitudinal angle of incident to reflected direction $(\phi_i - \phi_o)$ is important (see Figure 2.2). Therefore, the BRDF of isotropic materials can be reduced to three dimensions. Another simplification can be made for diffuse surfaces because they are hardly view-dependent. We assume that all incident light is equally distributed in all directions resulting in a constant BRDF at a surface point, which can easily be modulated by a texture. Such a surface is called *Lambertian*. Most walls and wallpapers of indoor environments are very close to be Lambertian.

An important property of the BRDF is the *Helmholtz law of reciprocity*, which states that the BRDF is independent of the light-flow direction:

$$f_r(x,\vec{\omega}_i,\vec{\omega}_o) = f_r(x,\vec{\omega}_o,\vec{\omega}_i) \tag{2.12}$$

This is a fundamental law most global illumination algorithms make use of, since it enables to trace light paths in both directions: from the eye and from the light source.

Because of the high-dimensionality of the BRDF (4D) a lookup table of real measured data can be quite large and inefficient. On the other hand, many materials have low frequency BRDFs with a simple shape that can be compressed using some hemispherical basis functions. Often a BRDF can be decomposed into two components: a diffuse part which is almost equal for all directions and a (narrow) specular lobe near the mirror direction of the view point. This observation leads to the rise of several analytical BRDF models that approximate (e.g. matching the model's parameters using least squares optimization) the measured data. This is a very common procedure and has the advantage of low storage costs and easier *PDF* approximation for importance sampling the BRDF. Since most analytical BRDF models are separable to a simple diffuse component and a more complex glossy component, it is also easier to compute the reflected radiance: the integral of the rendering equation, which we will describe next, can easily be split in two separate computations. This concept is exploited in all efficient algorithms in particularly in *radiosity*-based finite element algorithms and *photon mapping* [33].

2.2.2 The Rendering Equation

The rendering equation introduced by Kajiya [37] is the basis for all global illumination methods and will appear frequently further in the thesis. It is used to compute the outgoing radiance L_o at any point in the scene model. Thereby, L_o is interpreted as the sum of self-emitted radiance L_e and the reflected radiance L_r. It is defined as:

$$\begin{aligned} L_o(x,\vec{\omega}_o) &= L_e(x,\vec{\omega}_o) + L_r(x,\vec{\omega}_o) \\ &= L_e(x,\vec{\omega}_o) + \int_\Omega f_r(x,\vec{\omega}_i,\vec{\omega}_o) L_i(x,\vec{\omega}_i)(\vec{\omega}_i \cdot \vec{n}) d\vec{\omega}_i, \end{aligned}$$

where L_i is the incident radiance from a certain direction in the hemisphere. Unless only the direct illumination from the light sources is computed, we cannot reformulate this integral to a smaller integration domain, since, at least for closed environments, $L_i(x,\vec{\omega}_i)$ has a nonzero value for all pairs $(x,\vec{\omega}_i)$. Therefore, a natural way to solve the integral is Monte Carlo sampling (summarized in Section 2.3) using an appropriate hemispherical *PDF* $p(\vec{\Psi})$ to generate N random directions $\vec{\Psi}_i$. This results in the estimator:

$$L_o(x,\vec{\omega}_o) = L_e(x,\vec{\omega}_o) + \frac{1}{N}\sum_{i=1}^{N} \frac{f_r(x,\vec{\Psi}_i,\vec{\omega}_o) L_i(x,\vec{\Psi}_i)(\vec{\Psi}_i \cdot \vec{n})}{p(\vec{\Psi}_i)}. \tag{2.13}$$

In order to compute this estimator, N directions are generated from the *PDF* $p(\vec{\Psi})$. The BRDF and cosine term also need to be evaluated. A ray is traced in direction $\vec{\Psi}_i$ where we compute the outgoing radiance in direction $-\vec{\Psi}_i$ from the closest ray intersection point. In terms of variance reduction it is crucial to use a good *PDF* that is as close as possible to the incident hemispherical radiance function.

As one can easily see, the rendering equation is a recursive function because the radiance value is on both sides of the equation. One way to stop the recursive evaluation is to apply *Russian roulette* where the local hemispherical reflectance is taken as an appropriate absorption probability. Another (faster) way to compute the recursive integral is to store an approximative solution of the irradiance in the scene model and use it to compute the outgoing radiance in a second pass rather than using recursive ray tracing. A popular method is *photon mapping* which we will describe in Chapter 4.

It would be inefficient to compute the full reflected radiance via pure hemispherical *Monte Carlo ray tracing*, since the incident radiance field as well as the BRDF contains high frequencies which result in stochastic noise. Therefore, equation 2.13 is normally split into a sum of disjoint light contributions for different BRDF components that are solved individually and summed at the end. This is possible because light is additive and most BRDF models are separable into a diffuse and a glossy part. Such an approach is described in Section 4.4.

One important contribution to the reflected radiance is the direct illumination from visible light sources. Since only a few directions in the estimator 2.13 result in a nonzero contribution to the direct light, L_r is split into direct and indirect light from which only the latter one is computed by the integral over hemispherical directions. However, for the direct light as for all finite element algorithms we transform the integral in equation 4.2 to an integral over surface locations. This is done by replacing the differential solid angle:

$$d\vec{\omega}_i(x) = \frac{(\vec{\omega}_i \cdot \vec{n}_i)dA_i}{\|x - x_i\|^2}, \tag{2.14}$$

where the index i denotes another surface with differential area dA, normal $\vec{n}_i$, and position x_i. Remember the solid angle is the projection of the surface area onto the unit hemisphere. This leads to the rendering equation computed as the integral over all surface points:

$$L_o(x, \vec{\omega}_o) = L_e(x, \vec{\omega}_o) + \int_S f_r(x, x_i \to x, \vec{\omega}_o) L_o(x_i \to x) V(x, x_i) G(x, x_i) dA_i, \tag{2.15}$$

where $V(x, x_i)$ is the visibility function:

$$V(x, x_i) = \begin{cases} 1 & x_i \text{ and x are mutually visible} \\ 0 & \text{otherwise,} \end{cases} \tag{2.16}$$

$G(x, x_i)$ is called the geometry term comprising distance and orientation of two differential surfaces:

$$G(x, x_i) = \frac{(\vec{\omega}_i \cdot \vec{n}_i)(\vec{\omega}_i \cdot \vec{n})}{\|x - x_i\|^2}, \tag{2.17}$$

and S is the set of all surface points from which we want to compute the radiance contribution to point x. In case of the direct light computation it is obvious that equation 2.15 is more appropriate since the area of all light sources is usually relatively small and covers only a small solid angle on the hemisphere around x.

2.3 Monte Carlo Basics

In order to solve the rendering equation 2.13 by means of Monte Carlo ray tracing, one needs to know some theory of Monte Carlo integration. We will not introduce the basics of probability theory here but assume the reader to be familiar with it. This section is only a summary of Monte Carlo integration and variance reductions techniques. More information can be found for example in Kalos and Whitlock [38], Hammersley and Handscomb [24], in the thesis of Veach [83], and in Szirmay-Kalos's script [48]. Monte Carlo integration is a powerful technique that can handle arbitrary functions and multi-dimensional integrals at the same time. Let us consider the integral for the reflected irradiance in the rendering equation expressed in spherical coordinates (θ, ϕ):

$$\begin{aligned} L_r(x, \omega_o) &= \int_\Omega f_r(x, \omega_i, \omega_o) L(x, \omega_i) \cos\theta_i d\omega_i \\ &= \int_0^{2\pi} \int_0^{\pi/2} f_r(x, \theta, \phi, \omega_o) L(x, \theta) \cos\theta \sin\theta d\theta d\phi, \end{aligned} \tag{2.18}$$

since $d\omega_i = \sin\theta d\theta d\phi$. The estimator using Monte Carlo integration becomes:

$$\langle I \rangle = \frac{1}{NM} \sum_{i=1}^{N} \sum_{j=1}^{M} \frac{f_r(x, \theta, \phi, \omega_o) L(x, \theta_i, \phi_j) \cos\theta_i \sin\theta_i}{p(\theta_i, \phi_j)}, \tag{2.19}$$

where $p(\theta, \phi)$ is a 2D probability density function over the hemisphere.

Basically, Monte Carlo estimation works in the following way:

1. sampling according to a probability density function (PDF),
2. evaluation of the function at that sample,
3. and averaging the weighted values.

2.3.1 Variance Reduction

One problem with "blind" Monte Carlo integration is the slow convergence of the estimator to the correct solution. As the number of samples (N) increases, the variance of the Monte Carlo estimator decreases linearly with N. However, the error in the estimator is proportional to the standard deviation σ. Thus σ decreases with rate $\sqrt{N}$. To achieve faster convergence various methods have been developed, which we will briefly describe next.

2.3.1.1 Importance Sampling

The difficult part is the sampling of the probability density function. First of all, a "good" PDF p has to be found. A PDF is "good" if it resembles the function to be integrated (f). Intuitively, sampling a PDF can be understood as placing more samples in important regions of the domain where the function is greater since those regions contribute more to the integral resulting in a larger error. Because of that any knowledge about f is helpful. In the simplest case p is chosen to be uniform, which means that the integration domain should be equidistantly sampled. This is the best choice if no knowledge about f is given because if p is chosen badly (i.e. very different from f) the convergence of the estimator gets worse! In the ideal case the density p is proportional to f which completely eliminates the error:

$$p_{opt}(x) = \frac{f(x)}{\int_a^b f(x)dx} \Rightarrow \langle I \rangle = \frac{1}{N} \sum_{i=1}^{N} f(x_i)/p_{opt}(x_i) = \int_a^b f(x)dx \tag{2.20}$$

Paradoxically, this would require the knowledge of the quantity $\int f$ we want to estimate. Therefore, in practice, any function whose shape resembles f to a greater degree, is preferable to uniform sampling. It is also important that the *PDF* can easily be evaluated and integrated to build the *CDF* analytically.

To sample a *PDF* p we first need to compute the *cumulative distribution function* (*CDF*) by integrating p: $F(y) = \int_{-\infty}^{y} p(x)dx$, with $0 \leq F(y) \leq 1$ or in the discrete case with a domain beginning with a: $F(y) = \sum_{i=a}^{y} p(y)$. Next, the inverse of F is computed. This allows to draw samples ξ that are distributed according to p by simply generating uniform random values u: $\xi = F^{-1}(u)$.

In case that the *PDF* is too complex, rejection sampling can be applied. It increases the dimension of the function by one. For instance, in case of a one-dimensional function two uniform random variables (u_1, u_2) are needed, u_1 samples the domain (x) and u_2 is uniformly distributed in the maximum range of the *PDF* ($0 \leq u_2 < max(p)$). Sample u_1 is accepted if $u_2 < p(u_1)$ and rejected otherwise. However, rejection sampling is not very efficient if the *PDF* contains high peaks resulting in many rejected samples in the rejection area above the *PDF*. Because the dimension is increased by one, "standard" sampling from the inverse *CDF* is preferable. More efficient rejection sampling techniques have been proposed that try to reduce the rejection area by using a "tighter" shape on top of the *PDF* and hence increase the acceptance probability which is proportional to the efficiency.

Another possibility to generate samples from a probability distribution is to build a discrete *CDF* in form of a look-up table containing summed up precomputed *PDF* values. This representation is costly and gives only a piecewise constant approximation to the "real" *PDF*. For instance, this is a common way if we are dealing with measured data (e.g. local photon/importon distribution, BRDF data).

Importance sampling becomes indispensable for sampling of skewed functions such as glossy BRDFs. Often a BRDF is modelled by a simple analytical function that can even be used as a *PDF* itself (e.g. cosine power lobe). Most BRDF models are composed of a diffuse component $f_d = \frac{\rho_d}{\pi}$ and a glossy/specular component f_s, each sampled according to a different *PDF*. For the diffuse component one usually chooses a uniform *PDF* since the whole hemisphere contributes equally to the diffuse BRDF. Apart from the BRDF one can exploit one more fact hidden in the transformed rendering equation shown in 2.18. The incident radiance along a ray is weighted by the cosine of its azimuth angle. In addition, the sine of the azimuth angle is included as well due to the transformation from the integral over the solid angle to the spherical coordinates sampled from a unit square. This is why it is recommended to sample the diffuse BRDF according to the following distribution:

$$p_c(\theta, \phi) = \frac{\cos\theta \sin\theta}{\pi} = \frac{1}{2\pi} \cdot 2 \sin\theta \cos\theta, \tag{2.21}$$

where π in the denominator is due to the normalization of the *PDF* to integrate to 1. Since ϕ and θ are independent, p_c is separable to density functions for ϕ and θ with the corresponding *CDFs*:

$$\begin{aligned} F_1(\phi) &= \int_0^{\phi} \frac{1}{2\pi} d\Phi = \frac{\phi}{2\pi} \text{ and} \\ F_2(\theta) &= \int_0^{\theta} 2 \cos\theta \sin\theta d\Theta = \sin^2\theta. \end{aligned} \tag{2.22}$$

Both *CDFs* are easily invertible. After replacing $F_1(\phi)$ by a uniform random variable u_1 and $F_2(\theta)$ by u_2 we obtain: $\phi = 2\pi u_1$ and $\theta = \sin^{-1}\sqrt{u_2}$. Using p_c for Monte Carlo sampling of the rendering equation with a diffuse BRDF the estimator in 2.19 becomes:

$$\langle I \rangle = \frac{1}{NM}\frac{\rho_d(x)}{\pi}\sum_{i=1}^{N}\sum_{j=1}^{M}\frac{L(\theta_i,\phi_j)}{p_c(\theta_i,\phi_j)}\cos\theta_i\sin\theta_i = \frac{1}{NM}\rho_d(x)\sum_{i=1}^{N}\sum_{j=1}^{M}L(\theta_i,\phi_j) \tag{2.23}$$

This simple example shows the application of importance sampling in the rendering equation. This is the best way to sample a diffuse BRDF. For sampling of specular and glossy BRDFs several approaches have been proposed for various models [92, 59]. So far we have only spoken about BRDF sampling. Another strategy is sampling according to the incoming illumination (e.g. light source sampling [72]). An optimal importance sampling strategy requires the *PDF* to be proportional to the product of the incoming illumination and the cosine weighted BRDF. Unfortunately this is not trivial. First, the incoming indirect illumination part is not known during the random walk. And second, the combination of (direct) illumination sampling and BRDF sampling is quite difficult in practice. For the first problem, two approaches have been proposed: storing an approximative illumination representation in a preprocessing phase, e.g. photon map [34], or adapting the sampling to the history of previous results, e.g. 5D adaptive tree [44].

2.3.1.2 Multiple Importance Sampling

The problem of combining illumination and BRDF sampling can be dealt with a linear combination of the two estimators by choosing appropriate weights that add up to one and are inversely proportional to the variance of their corresponding estimator. For example consider the rendering equation for a ideal specular BRDF (i.e. dirac impuls). The variance for the BRDF estimator is zero and thus its weight should be one whereas the weight for the incident radiance estimator becomes zero. In this case we actually only need to sample the BRDF in mirror direction. However in general it is not so simple and the variance is difficult to estimate in advance. Consequently, local importance sampling strategies are often restricted to either use the cosine weighted BRDF or the incoming illumination to identify important directions. A robust strategy for *multiple importance sampling* was introduced by Veach [83]. He uses the *balance heuristic* to determine the weights of individual samples from different estimators. For m individual estimators with *PDF* p_i and n_i samples the combined estimator F using the balance heuristic becomes:

$$F = \frac{1}{N}\sum_{i=1}^{m}\sum_{j=1}^{n_i}\frac{f(X_{i,j})}{\sum_k n_k/Np_k(X_{i,j})}, \tag{2.24}$$

where $N = \sum_{i=1}^{m} n_i$ is the total number of samples. Note that the weights in the denominator add up to one and the balance heuristic results in an unbiased estimator. Its variance is equal to the "optimal" estimator plus an additive error term.

2.3.1.3 Control Variates

Importance sampling is one way to reduce the variance in the estimate. Yet another possibility are *control variates*. If a similar function g to the original function f to be integrated can be found and if g can be integrated analytically, then we could reduce the variance by:

$$I = \int f(x)dx = \int g(x)dx + \int f(x) - g(x)dx. \tag{2.25}$$

Since $\int g(x)dx$ is analytically solvable, this boils down to estimate $\int f(x) - g(x)dx$. Control variates are very effective and should be used if $f(x)/g(x)$ is almost constant.

2.3.1.4 Low Discrepancy Sampling

Last but not least, two important variance reduction techniques need to be mentioned: *stratified sampling* and *Quasi-Monte Carlo sampling*. Both techniques reduce variance by avoiding the clumping of samples. Stratified sampling divides the domain into M disjoint sub-domains called strata each evaluating the integral separately with one or more uniformly distributed samples. Quasi-Monte Carlo techniques replace randomness entirely by deterministic sequences called low-discrepancy sequences (LDS) in order to minimize the *discrepancy* (measure for the clumping of samples) on the expense of introducing correlation between samples. Many LDS based on Quasi-Monte Carlo techniques have been developed. Popular ones are *Halton*, *Hammersley*, *Niederreiter*, and *Sobol*. The results converge as fast as for stratified sampling without the dependence on the number of samples. Furthermore, we inherently get stratification over several dimensions. On the other hand, Quasi-Monte Carlo sampling can lead to visible artifacts due to deterministic sampling patterns. Therefore, a combination of randomness and LDS, which is called *randomized Quasi-Monte Carlo* (RQMC) is sometimes preferable. More details about the Quasi-Monte Carlo method can be found in [60, 41].

2.4 Global Illumination Algorithms

In this section we give a summary of the most popular methods aiming at solving the rendering equation. In general, rendering methods can be classified to *finite element algorithms* or *random walk algorithms*. Finite element methods approximate the *infinite*-dimensional function space by a *finite*-dimensional function space defined by a set of basis functions. Finite element methods have been successfully applied to the diffuse global illumination problem and are referred to as *radiosity* methods. The second class of random walk algorithms can be further specified to *shooting-type* algorithms that start from the light sources, *gathering-type* algorithms that start from the eye, and *bi-directional* (multi-pass) algorithms which exploit both types. A profound description of global illumination algorithms can be found in [17, 48].

However, not all algorithms yield a complete global illumination solution. In order to describe the possible light transport an algorithm is able to simulate, we will use some light transport notation.

2.4.1 Simplified Light Transport Notation

When it comes to the classification of a rendering algorithm or when we want to split the integral of the rendering equation 2.13 into separate components each solving a different light contribution, it is useful to have a formal light path notation. Heckbert [30] has introduced a compact description, where each light path vertex can be of the type:

- **L** point on a light source
- **E** location on the eye (or camera)
- **D** diffuse reflection

- **S** specular reflection
- **G** glossy reflection

In reality a path always starts on a light source but in computer graphic algorithms it is preferable to allow both directions: either from the light source **L** or from the eye **E**. In between can be any combination of reflections which is described by using regular expressions:

- $(r)+$ at least one reflection of type r
- $(r)*$ zero or more reflections of type r
- $(r)?$ at most one reflection of type r
- $(r|r')$ either a reflection of type r or r'

A path $L(S|G)+DE$ for example represents a caustic path that starts at the light source and has one or more specular (or glossy reflection) before being reflected at a diffuse surface towards the eye.

2.4.2 Finite Element Radiosity Techniques

The classical radiosity method is the most well-known and one of the earliest approaches for solving the rendering equation. It is a finite element method and solves the rendering equation by a discretization of the integral into a system of linear equations. A significant amount of research has been devoted to radiosity techniques, in order to make them efficient and more adaptive. A profound introduction would be out of the scope of this thesis. We will only give a rough overview and derivation of the mathematical foundation since it might help in understanding the problem of global illumination and the relation to the photon mapping concept. A good description of the method can be found in [11, 12].

Radiosity algorithms rely on a mathematical simplification of the problem. The first simplification is the restriction to surfaces in the model that are Lambertian (i.e. have constant BRDF denoted by $f_{r,d}(x) = \frac{\rho_d(x)}{\pi}$). This allows to replace the directionally invariant radiance $L(x)$ in the rendering equation 2.15 by the radiosity (or radiant exitance) $B(x) = \pi L(x)$

$$\begin{aligned}
L(x) &= L_e(x) + \int_S f_{r,d}(x) L(y) V(x,y) G(x,y) dA_y && | \; f_{r,d}(x) := \rho_d(x)/\pi \\
\Rightarrow L(x) &= L_e(x) + \frac{\rho_d(x)}{\pi} \int_S L(y) V(x,y) G(x,y) dA_y && | \; *\pi \quad | \; \pi L(x) := B(x) \\
\Rightarrow B(x) &= B_e(x) + \rho_d(x) \int_S \frac{B(y)}{\pi} V(x,y) G(x,y) dA_y
\end{aligned} \tag{2.26}$$

The second simplification is the discretization of the scene surfaces to a finite set of N (small) patches each with constant radiosity B_i. The average radiosity B_i for a patch i is then computed by integrating over the patch area A_i

$$B_i = \frac{1}{A_i} \int_{A_i} B(x) dx. \tag{2.27}$$

Now we can combine the two results by plugging equation 2.26 into equation 2.27 and assuming constant reflectivity $\rho_d(x) = \rho_i, \forall x \in A_i$ for each patch, which results in

$$\begin{aligned} B_i &= \frac{1}{A_i}\int_{A_i} B(x)dx \\ &= B_{e,i} + \frac{1}{A_i}\int_{A_i}\rho_d(x)\sum_j^N \int_{A_j}\frac{1}{\pi}B(y)V(x,y)G(x,y)dydx \\ &= B_{e,i} + \sum_j^N \rho_i B_j \frac{1}{A_i}\int_{A_i}\int_{A_j}\frac{1}{\pi}V(x,y)G(x,y)dydx \\ &= B_{e,i} + \rho_i \sum_j^N F_{ij}B_j, \end{aligned} \tag{2.28}$$

where the first term $B_{e,i}$ is the average self-emitted radiosity of patch i and the second term represents the irradiance received from all N patches of the model. The term F_{ij} is called the form factor and can be considered as a discrete probability density function where F_{ij} corresponds to the probability that a photon emitted from patch i lands on patch j. The computation of form factors F_{ij} between any two patches i and j is a four-dimensional integral which is expensive to evaluate for thousands of patches in the model since it has quadratic complexity and needs to evaluate the visibility term between any two patches. The form factor is defined as

$$F_{ij} = \frac{1}{A_i}\int_{A_i}\int_{A_j}\frac{V(x,y)G(x,y)}{\pi}dydx. \tag{2.29}$$

The form factor is the fundamental concept of radiosity methods and obeys the following rules:

1. $\forall i, j.0 \leq F_{ij} < 1$, no negative patches possible,
2. $A_iF_{ij} = A_jF_{ji}$, reciprocal when discarding the area of the patches,
3. $N_{ij}/N_i \approx F_{ij}$, the ratio of the number of received photons N_{ij} on patch j to the number of emitted photons N_i from patch i corresponds to the form factor F_{ij},
4. $\sum_j^N F_{ij} = 1$, no energy gets lost in closed environments (i.e. all emitted photons hit another patch j)
5. $P_i = A_iB_i \Rightarrow P_i = A_iB_{e,i} + \rho_i\sum_j^N A_iF_{ij}B_j = P_{e,i} + \rho_i\sum_j^N F_{ji}A_jB_j = P_{e,i} + \rho_i\sum_j^N F_{ji}P_j$, the radiosity equation expressed in terms of power P: the power emitted from patch i corresponds to the self-emitted power $P_{e,i}$ and the reflected power received from all other patches j. Here, F_{ji} is reversed and represents the fraction of the power from patch j arriving at patch i.

The classical radiosity equation in 2.28 boils down to a system of linear equations, which can be solved by an iterative method such as Jacobi or Gauss-Seidel for example.

The radiosity method is based on a nice mathematical description and has the advantage of view-independence. Once the form factors are computed for each pair of patches and the radiosity is computed at each patch, it can be used to efficiently compute a camera walk through the model. Therefore, its main application is the realistic rendering of indoor environments in

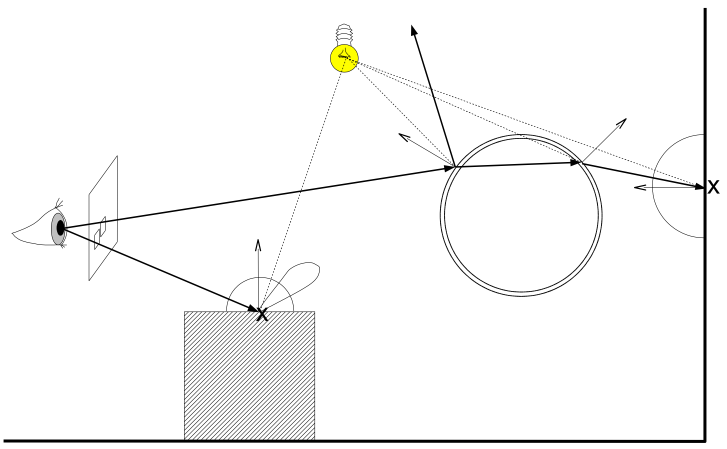


Figure 2.3: *Classic ray tracing: a deterministic algorithm that computes the direct illumination and indirect illumination from ideal reflections or ideal transmissions.*

architectural models. On the other hand, a discretization of the scene model is often difficult and must deal with the adaptation to shadow boundaries and illumination gradients. Furthermore, only Lambertian surfaces (i.e. diffuse light transport of type $L(D) * E$) can be handled and the combination with other methods is poor. This is one reason why the classical radiosity algorithms have lost the attention in current research.

2.4.3 Classic Ray Tracing

Classic ray tracing was introduced to computer graphics by T. Whitted in 1980 [96]. It exploits the fact that radiance is constant along a line in empty space and the BRDF is reciprocal. Hence, we can reverse the computation by tracing rays from the camera into the scene rather than photons from the light, which have little probability to reach the observer. This allows to render sharp shadows and specular surfaces efficiently. The simplest method of this class is *ray casting* which only computes the direct illumination from primary rays ($E(D|G|S)?L$). Ray casting is easily extended to *recursive ray tracing*, which can compute multiple light bounces of ideal reflection and refraction. However, we are limited to direct illumination at any point on the path. Classic recursive ray tracing is only capable of computing light paths of the type $E(S) * (D|G)?L$, which represents all path from the light source to the eye that hit at most one diffuse or glossy surface before they are specularly reflected (arbitrarily often) towards the eye. If the eye is represented by a point and the surfaces are perfectly specular, then the probability that a photon emitted by a light source reaches the eye becomes zero. Consequently, this kind of light paths is best solved by recursive ray tracing starting from the eye. Nevertheless, recursive ray tracing is a deterministic algorithms that can only handle perfectly specular BRDFs and point light sources. Therefore, it cannot compute full global illumination.

An extension of recursive ray tracing to obtain a global illumination solution is the *distribution ray tracing* algorithm suggested by Cook [13], which uses stochastic sampling to handle effects as motion blur, soft shadows, and depth of field. Distribution ray tracing uses *Monte Carlo* integration and can therefore solve the rendering equation.

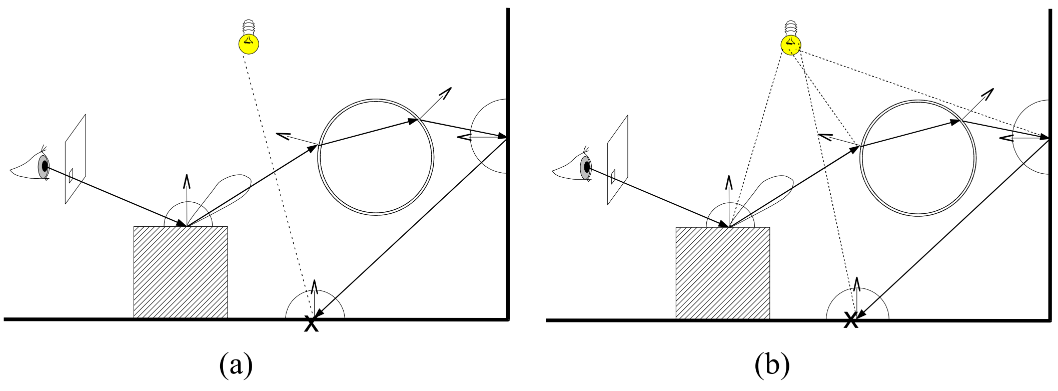


Figure 2.4: *Path tracing: computing full global illumination via random gathering walks starting at the eye. The two typical methods are: (a) trace a random path and combine direct light and eye path only at the end of the eye path (i.e. when stopped by Russian roulette) via one or more shadow rays to the light source(s), (b) compute the direct illumination at each eye path vertex.*

2.4.4 Monte Carlo Ray Tracing

Monte Carlo ray tracing based algorithms are probably the most general global illumination methods, because they do not require any precomputation or simplification of the model. Monte Carlo ray tracing reduces the problem of illumination computation to tracing rays through the model and computing radiance along those. This way we can handle arbitrary topology (e.g. procedural geometry) and any type of BRDF. The memory consumption is negligible and the results are unbiased. However, the results converge very slowly to a "noise-free" solution due to the fact that the standard error in Monte Carlo integration is proportional to $1/\sqrt{N}$. We will summarize the most popular random walk algorithms but discard *shooting-type* algorithms that start from the light sources. Shooting-type algorithms can be considered as the inverse of the classic ray tracing and path tracing algorithm. An exhaustive survey of Monte Carlo methods is given in the thesis of Veach [83].

2.4.4.1 Path Tracing

A more general version of the classic ray tracing is *path tracing.* Path tracing makes it possible to compute a complete unbiased global illumination solution. It can simulate all possible light paths: $E(S|D|G) * L$ and is therefore often used as a reference solution for comparison with other more efficient (but biased) rendering algorithms. Path tracing was introduced by Kajiya in 1986 [37] as a solution to *his* proposed rendering equation [37]. Path tracing is a straightforward extension to classical ray tracing. An important difference to distribution ray tracing is that it only uses one reflected ray to estimate the indirect illumination. This ensures that the number of primary rays is the same as for the reflected rays and we avoid the exponential growth in the number of rays with increasing reflections. Path tracing makes use of *Russian roulette* to stop the random walk and sample diffuse as well as specular or glossy surfaces in proportion to their maximum reflected energy. For each reflection event a stochastic decision is made using the surface properties, whether to continue by sampling a new direction or to stop. The samples should be concentrated towards bright regions and the mirror direction of specular BRDFs. The sampling direction is crucial because it strongly influences the variance of the estimate.

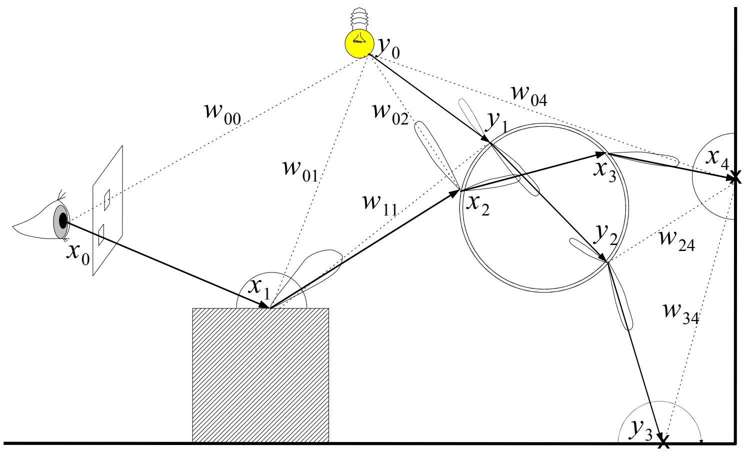


Figure 2.5: *Bidirectional path tracing: combining light path ($y_0, \ldots, y_3$) and gathering path ($x_0, \ldots, x_4$) in* bidirectional path tracing. *The dotted lines represent all possible (non-occluded) links between both paths. The weights of the links are denoted by w_{ij}. The paths can either be combined at only* one *link according to the probability of the resulting path or as a weighted sum of all possible combinations using multiple importance sampling. For example weight w_{11} and weight w_{24} should have a very low probability due to the deviations from the glossy reflection, whereas a higher probability should be assigned to w_{02} and w_{34}.*

Since the probability for a path to hit a light source is low for small light sources, one either shoots one or more shadow rays at *each* reflection event or only at the end of the random walk (absorption by Russian roulette). To understand why this method works, it is necessary to comprehend the concept of *Monte Carlo integration* (see Section 2.3). The only problem with path tracing is variance in the estimates that is seen as noise in the final image. To compute an accurate estimate for a pixel it is usually necessary to average the result of many paths. By averaging a large number of sample rays for a pixel, we get an estimate of the integral over all possible light paths through that pixel. Often, thousands of rays per pixel are necessary to obtain an acceptable quality for the rendered image. This reveals the problem of simple path tracing: it neither accounts for knowledge about the scene and illumination nor does it reuse computed information from neighboring paths. Each path is computed individually. Therefore, path tracing works poorly in case of high frequency indirect illumination such as caustics. To make path tracing practical, efficient adaptive sampling techniques are necessary that sample ”important” paths more densely than paths with little contribution to the pixel. Such techniques are briefly discussed in Section 2.3.

2.4.4.2 Bidirectional Path Tracing

Bidirectional path tracing was introduced in 1993 by Lafortune and Willems [45] as an extension to the path tracing algorithm. The algorithm traces paths starting from the eye as well as the light sources and combines the information from both paths. The vertices along the light and eye path are connected via shadow rays and the final pixel estimate is computed as a weighted sum of all path combinations. However, it is also possible to sample only one combination. The choice of the weights has a great influence on the variance of the combined estimate. By setting all those path weights to zero, which include light path segments, we obtain standard path tracing. A common choice for the weights is the power heuristic that sets the weights according to the probability density of a path segment, which is proportional to the BRDF sampling probability times the inverse of the squared distance between the two path vertices times their orientation with respect to their normals (as in Equation 2.17). The weights can be defined by various schemes. However, one necessary condition is that the weights corresponding to paths of the same length must sum up to one. Bidirectional path tracing performs normally better than standard path tracing especially when the light sources are small or the image contains caustics.

2.4.4.3 Metropolis Light Transport

The *Metropolis Light Transport* technique (MLT) was introduced by Veach and Guibas [84] in 1997 as a method for exploiting the knowledge of the path space more efficiently. The key idea of MLT is that paths are sampled according to their contribution to the final image. Metropolis sampling was first introduced by Metropolis et al. [54] in 1953. The concept of Metropolis sampling is different from stochastic path tracing. Instead of randomly sampling a function f to compute the integral, the Metropolis method generates a sequence of samples distributed according to the unknown function f. This results in a concentration of path in bright regions of the image. First, MLT starts with random bidirectional path tracing of the space of all paths in the model. These paths are then randomly cloned and mutated using several strategies: **bidirectional mutations** randomly replace segments of a path to ensure the entire path space is visited and the results are unbiased; **pertubations** try to make small changes to a path by moving only a few vertices for example in order to focus on specific illumination effects such as caustics. A mutated path y is accepted based on an acceptance probability $a(y|x)$:

$$a(y|x) = min\left\{1, \frac{f(y)T(x|y)}{f(x)T(y|x)}\right\}. \tag{2.30}$$

Here $a(y|x)$ is the acceptance probability of the path y given path x, $f(y)$ is the resulting radiance from path y, and $T(y|x)$ is the transition probability density function for the mutation from path x to path y. Given a uniform random number ξ, we decide whether we take the new mutated path y ($\xi < a(y|x)$) or keep the old path x. The mutated paths are then distributed according to the radiance. MLT is an unbiased technique and is efficient at computing difficult illumination situations where few regions in the scene are responsible for most of the illumination (e.g. light ports such as holes, windows) and caustics. For scenes where the entire space is equally important for the indirect illumination, MLT is not much more efficient than bidirectional path tracing. Furthermore, MLT is quite complicated to implement and the right choice of the mutation strategies is highly scene-dependent. For instance, it is possible that mutations will result in slower convergence when exploring only a few of many important paths.

2.4.5 Photon Mapping

Photon mapping is a two-pass algorithm introduced by Henrik Wann Jensen [35, 33]. Like bidirectional path tracing, it traces illumination paths from the eye and from the light sources but stores the information (the hit points) of the light paths in the first pass. In the second pass this information is reused by all eye path vertices in the neighborhood. The illumination can then be computed from the density of the neighboring light vertices (photons). Photon mapping is a biased technique which trades bias with variance since it performs a convolution of the irradiance

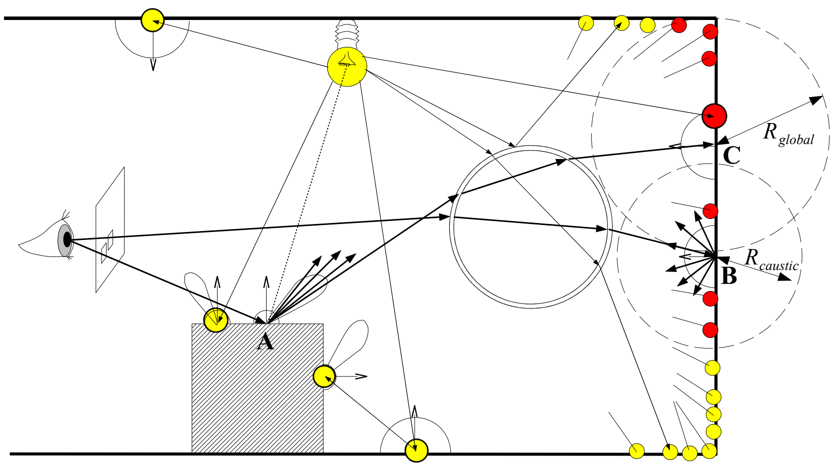


Figure 2.6: *Photon mapping: all photons (discs) are stored in the global map, which is first queried after final gathering (point C). Caustic photons (small discs) are stored separately in the caustic map. All photons are only stored when they hit a diffuse or glossy surface. At point A and point B final gathering via BRDF importance sampling is applied to compute the indirect illumination from the photon map. At point B the caustic map is queried to compute the directly visible (here through 2 ideal reflections!) caustic indirect illumination. Note that photons of all light paths are included in a global map query (point C).*

(flux density) on a surface. Therefore, the photon map represents just an approximation to the real irradiance. Except for caustic paths $(L(S)+DE)$, it is commonly not used in a direct visualization of the irradiance. The photon map is queried after a prepended Monte Carlo sampling step called *final gathering* (see Figure 2.6). Photon mapping combined with Monte Carlo sampling can simulate all light paths $(L(D|G|S)*E)$ in an efficient way. Photon mapping is described in detail in Chapter 4. Its mathematical derivation is briefly discussed in the next chapter.

2.4.6 Instant Radiosity

Instant radiosity is an elegant two-pass method introduced by Keller [42] in 1997, which is related to bidirectional path tracing and photon mapping. The idea is simple though effective. The indirect diffuse illumination is computed by direct illumination from a set of photons that function as point light sources as shown in Figure 2.7. Therefore, a "good" distribution (low discrepancy) of the indirect point light sources is very important, which can be achieved with Quasi-Monte Carlo sampling. Instant radiosity can be efficiently implemented using graphics hardware for less complex scenes. However, its limitation is that it only computes diffuse indirect illumination since it is restricted to view-independent diffuse surfaces. Thus, the computed light transport is $L(D|G|S)*DD(S)*E$.

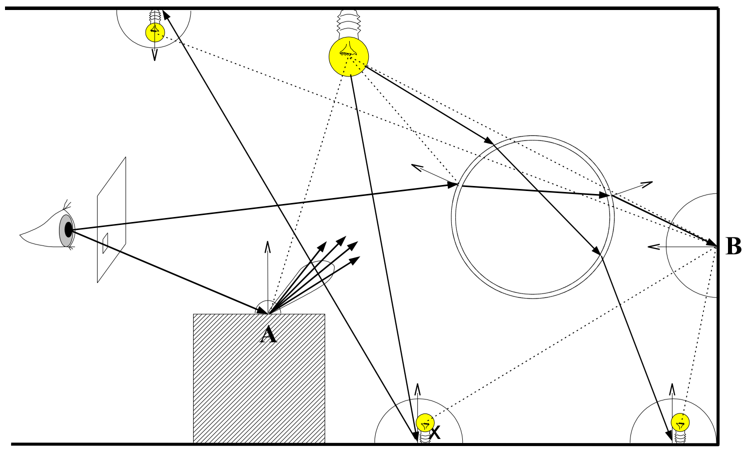


Figure 2.7: Instant Radiosity: *diffuse indirect illumination computation using* instant radiosity. *Individual photon hits on diffuse surfaces function as virtual point light sources. The illumination at point B is computed by direct illumination of primary light source and virtual point light sources, e.g. by tracing shadow rays (dotted lines). Note that the indirect illumination computed with instant radiosity needs to be view-independent. Thus, specular or glossy BRDFs (point A) should be sampled using Monte Carlo ray tracing (final gathering) before applying instant radiosity. A problem with instant radiosity is the geometric term in the rendering equation (Equation 2.15). It can take on any value due to the squared distance in the denominator (weak singularity). This happens for point light sources located in corners where the distance to the illuminated point can become arbitrarily small. Such cases can only be dealt with simple bias-introducing heuristics, e.g. culling if the distance of a point light source is below some small threshold.*

Chapter 3

Introduction to Statistical Density Estimation

This chapter is intended to give an overview and understanding of *statistical density estimation* and the involved difficulties. We will also present the most common methods used for density estimation. However, for a profound mathematical survey of density estimation we refer to standard text books on the topic [74] and [89].

3.1 Introduction

A fundamental concept in statistics is the probability density function (PDF). Given a random variable X with PDF f, one can associate probabilities with X to be computed by

$$P(a \leq X < b) = \int_a^b f(x)dx \text{ for all a < b.} \tag{3.1}$$

Intuitively it means that the probability to draw a sample in range a to b is given by the integral (i.e. area under the graph of f) of f in range $[a, b)$. This makes it clear that the probability becomes zero if a and b converge to a point. Given a PDF f we can draw a sample whose data points are distributed according to f.

Density estimation can be considered as the inverse approach. Now we are given a set of observed data points that are distributed according to an unknown PDF. Hence, the goal of density estimation is to estimate a probability density function $f(\cdot)$ of a random variable X. This is a common problem in many contexts of statistics where we are interested in a human readable visualization or a measurable function rather than a set of loose data points. Density estimation is widely used in pattern recognition and classification, in computer graphics and image processing, communication, signal processing and many other fields of computer science. A distribution is not necessarily univariate. It can have arbitrary dimension although in statistics the most attention is payed to the univariate case.

If a particular form of the density is assumed or known, then *parametric* estimation can be used. For example if the data samples are drawn from a normal distribution with mean μ and variance σ^2, then we only need to estimate these two parameters and substitute them into the formula for the normal density. On the other hand, *nonparametric* estimation is employed when nothing can be assumed about the shape of the distribution. In computer graphics especially in global illumination we are mostly concerned with nonparametric density estimation in 2D (bi-variate). However, for volumetric effects for instance, one must consider tri-variate density estimation. In the following sections we will have a look at a few common estimation methods. We will denote by $\hat{f}$ the density function estimate, h the *window width* or *bandwidth*, and x the density estimation point. We will assume that we are given a sample of n real observations $X_1, ..., X_n$ drawn from a *distribution function* f that we want to estimate.

3.2 Histograms

The *histogram estimator* is the oldest and simplest density estimation method. In its simple version it is non-adaptive and the domain is divided into a number of equal-sized bins with a piecewise constant approximation in each bin. The density at a point x is then computed as

$$\hat{f}(x) = \frac{1}{n \cdot h} \cdot (\text{ no. of samples in same bin as } x) \tag{3.2}$$

Histograms can only provide a coarse representation of the density function and it is difficult to find the globally optimal bin width h. In case of *adaptive histograms* the constant bin width h is replaced by a variable width h_i. Moreover, the density reconstruction also depends on the discretization of the domain. For example, a translation of the bins can change the density estimate. There are less biased estimators than histograms. Nevertheless, a histogram can be useful as a *pilot* or *plug-in* estimate since it has the least computational complexity. Histograms can be very efficiently implemented using *summed area tables* which allow the density estimate to be computed in $\mathcal{O}(1)$ time.

3.3 Kernel Density Estimation

Kernel Density Estimation (KDE) introduced by Rosenblatt (1956) and Parzen (1962), has been widely studied. It is the most common technique in statistics. It can be further classified to *parametric* and *nonparametric* density estimation. In parametric approaches, one assumes that the unknown function is of a particular family of distribution functions. KDE in form of nonparametric density estimation has become popular for visualization of univariate data but not so significant for multivariate data visualization due to numerical difficulties and computational costs. For KDE we do not have fixed bins as for histograms but *variable intervals* centered around the point in query and defined by the *bandwidth* h wherein we process the number of samples. Intuitively speaking, it can be formulated as:

$$\hat{f}(x) = \frac{1}{n \cdot 2h} \cdot (\text{no. of samples within } (x-h, x+h)) \tag{3.3}$$

In this naive case, all samples in the interval are weighted equally. To have a more general formulation, we can introduce a weighting function called *kernel* $\mathcal{K}$ that inherently counts the number of samples falling in the interval and additionally weights them individually according to their distance from x.

$$\hat{f}(x) = \frac{1}{n \cdot h} \sum_{i=1}^{n} \mathcal{K}\left(\frac{x - X_i}{h}\right) \tag{3.4}$$

In order to make $\mathcal{K}$ independent from the bandwidth h, the distance between x and X_i is divided by h and the result is scaled by $1/h$: $\mathcal{K}_h(t) = \frac{1}{h}\mathcal{K}\left(\frac{t}{h}\right)$. To simplify the scaled kernel expression, we denote it by $\mathcal{K}_h$. $\mathcal{K}_h$ stays invariant under integration regardless of h. K_h is also a PDF but its variance is $Var(\mathcal{K}_h) = Var(\mathcal{K}) \cdot h^2$. Hence, the smaller the bandwidth h the smaller the variance of K_h since its density concentrates about its mean, zero. The kernel $\mathcal{K}$ can be any symmetric non-negative function that satisfies the normalization condition:

$$\int_{-\infty}^{\infty} \mathcal{K}(x)dx = 1.$$

Since the density estimate $\hat{f}$ is a linear sum of the translated and scaled kernel function $\mathcal{K}_h$, it inherits $\mathcal{K}$'s continuity and differential properties. In case $\mathcal{K}$ is the box function, the result will be a piecewise constant step function. On the other hand, if we choose $\mathcal{K}$ to be the normal density (i.e. Gaussian) the resulting curve $\hat{f}$ will be smooth everywhere having derivatives of all orders.

3.4 K-Nearest Neighbors Estimator

The *k-nearest neighbors* (kNN) estimator is another important method for density estimation. It is perhaps the simplest adaptive method and can be efficiently implemented. Therefore, it is widely used for large data sets as in photon mapping. The key concept is that the amount of smoothing adapts to the local density, more precisely the bandwidth is inversely proportional to the density. Hence, the bandwidth is chosen as the distance $d_k(x)$ from x to the k-th data point where k is the furthest point from x. Then the naive density estimate becomes

$$\hat{f}(x) = \frac{k}{2 \cdot n \cdot d_k(x)} \tag{3.5}$$

The global degree of smoothing is controlled by a parameter k which requires k-median sorting of the nearest neighbors. The nearest neighbor estimate is not a smooth curve and has discontinuities in the first derivative because the distance function $d_k(x)$ does not have C^1 continuity properties. Furthermore, the kNN estimate can never be zero and suffers from "heavy tails" in regions where the density function f is (almost) zero (e.g. outside the domain) since the bandwidth $h = d_k(x)$ is increased until the desired number of points k has been found. We can also formulate a more general definition called the generalized nearest neighbor estimate including an arbitrary kernel

$$\hat{f}(x) = \frac{1}{n \cdot d_k(x)} \sum_{i=1}^{n} \mathcal{K}\left(\frac{x - X_i}{d_k(x)}\right), \tag{3.6}$$

where $\mathcal{K}$ is a kernel function. This formula is closely related to the KDE if we replace the constant bandwidth h by $d_k(x)$.

3.5 Variable KDE

The *variable kernel density estimation* is similar to the kNN estimation. It also adapts the amount of smoothing to the local density. However, in contrast to the kNN estimation, the adaptation is completely independent from the estimation point x. Instead of searching for the k nearest data points around x, we search for the k nearest points in the neighborhood of every data point X_i and associate the distance $d_{i,k}$ with the bandwidth of X_i. The variable kernel estimate is then defined by

$$\hat{f}(x) = \frac{1}{n} \cdot \sum_{i=1}^{n} \frac{1}{d_{i,k}} \mathcal{K}\left(\frac{x - X_i}{d_{i,k}}\right). \tag{3.7}$$

The variable KDE is superior to the kNN estimation since the estimate is a probability density function itself (provided $\mathcal{K}$ is one) with all the smoothness properties inherited by the kernel. It does not suffer from "heavy" tails since the bandwidth is independent from x. One drawback that should be mentioned is that the computation is more expensive than kNN estimation due to the individual bandwidth precomputation for each sample point X_i. This is in particular an issue if the number of observations $X_1, ..., X_n$ is much greater than the number of estimation points x.

3.6 Window Width and Kernel Choice

In this section we will give a theoretical introduction to density estimation for the kernel method which is the basis of all density estimation techniques.

The key issue with kernel methods is how the window width h is chosen. If it is too wide, the estimate will blur out relevant detail in parts of the domain with high density of samples, while if it is too narrow, the estimate will be too noisy in particular in the tails of the distribution where the density of points is sparse (see Figure 3.1).

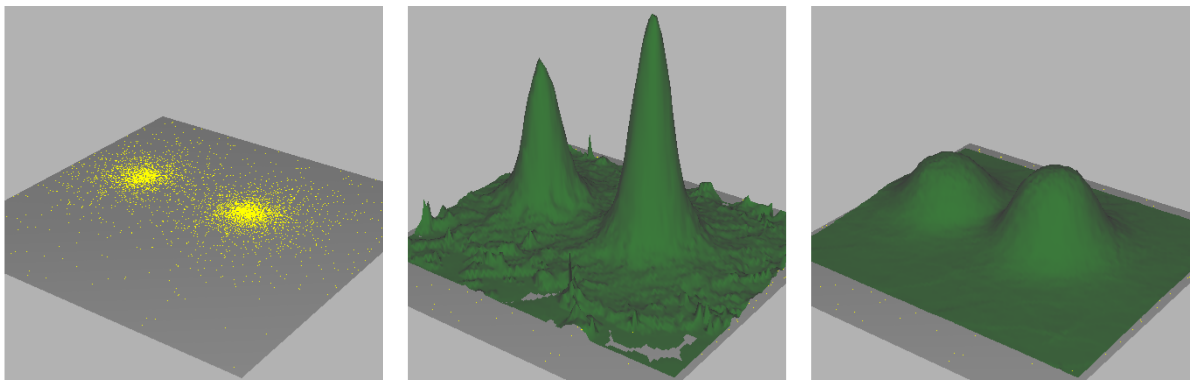


Figure 3.1: *The importance of the bandwidth in kernel density estimation: (left) The distribution of points drawn from two translated point light sources, (middle) the 2D (noisy) kernel density estimate using a small bandwidth ($R = 1$), (right) the (blurred) kernel density estimate using a large bandwidth ($R = 2$)*

This implies a trade-off between *variance* (random error) and *bias* (systematic error). Consequently, one might assume that the density estimate should comprise a minimum number of nearest samples. This intuitively leads to the assumption of coupling the bandwidth with the number of neighboring samples which is known as k-nearest neighbors (kNN) density estimation which we have already described. Soon it will become clear that this is not the best choice for density estimation and we will present some alternative methods later on.

In order to derive formulas for the optimal bandwidth and kernel, we will first describe a few measures of the error we can minimize introduced by the kernel methods. A typical measure of an estimator $\hat{f}$ is the *mean squared error* (MSE)

$$MSE(\hat{f}) = E(\hat{f} - f)^2 \tag{3.8}$$

which can be decomposed into a sum of *variance* and *squared bias*

$$MSE(\hat{f}) = Var(\hat{f}) + (E\hat{f} - f)^2. \tag{3.9}$$

We can show that the bias in KDE arises from a convolution with a kernel $\mathcal{K}_h$. Let us reformulate the expectation of our estimator $\hat{f}$ as follows

$$E\hat{f}(x,h) = E\left(\frac{1}{n}\sum_{i=1}^{n}\mathcal{K}_h(x - X_i)\right) = E\mathcal{K}_h(x - X) = \int \mathcal{K}_h(x-y)f(y)dy = (\mathcal{K}_h * f)(x), \tag{3.10}$$

where X is a random variable with density f. Thus, the expectation of $\hat{f}$ is simply a convolution of f with a kernel $\mathcal{K}_h$. This allows us to write the bias of $\hat{f}$ as

$$E\hat{f}(x,h) - f(x) = (\mathcal{K}_h * f)(x) - f(x). \tag{3.11}$$

In a similar way, we can express the variance as

$$Var\{\hat{f}(x,h)\} = E(\hat{f}^2) - (E\hat{f})^2 = \frac{1}{n}\{(\mathcal{K}_h^2 * f)(x) - (\mathcal{K}_h * f)^2(x)\}. \tag{3.12}$$

When we plug these results into the formula of the MSE, we obtain

$$MSE\hat{f}(x,h) = \frac{1}{n}\{(K_h^2 * f)(x) - (\mathcal{K}_h * f)^2(x)\} + \{(\mathcal{K}_h * f)(x) - f(x)\}^2. \tag{3.13}$$

Often it is preferable to estimate the error over the entire domain rather than a fixed point. Therefore, the *integrated square error* (ISE) is given by $ISE\{\hat{f}(\cdot,h)\} = \int\{\hat{f}(x,h) - f(x)\}^2dx$. However, more important is the *mean integrated square error* (MISE) since it measures the mean error over more than just one sample drawn from f

$$MISE\{\hat{f}(\cdot,h)\} = E[ISE\{\hat{f}(\cdot,h)\}] = \int E\{\hat{f}(x,h) - f(x)\}^2dx = \int MSE\{\hat{f}(x,h)\}. \tag{3.14}$$

We could now insert the expression for variance and for bias into the MISE equation. The resulting formula however, is fairly complicated to solve for the optimal bandwidth h_{opt}. Therefore, one usually uses an approximative but simpler expression for bias and variance derived from a *Taylor expansion* of the expected value of $\hat{f}$. First, we reformulate the estimation of $f(x)$ from Equation 3.10 by a change of variable $t = x - hy \Rightarrow dt = -hdy$:

$$E\hat{f}(x,h) = (\mathcal{K}_h * f)(x) = \int \frac{1}{h}\mathcal{K}\left(\frac{x-t}{h}\right)f(t)dt = \int \mathcal{K}(y)f(x-hy)dy \tag{3.15}$$

Expanding $f(x - hy)$ in a Taylor series at x we obtain

$$f(x-hy) = f(x) - hyf'(x) + \frac{1}{2}h^2y^2f''(x) + o(h^2). \tag{3.16}$$

Let us assume we have a symmetric kernel function that obeys the following properties

$$\int \mathcal{K}(z)dz = 1, \text{ (PDF) } \int z\mathcal{K}(z)dz = 0, \text{ (symmetry) and } \int z^2\mathcal{K}(z)dz < \infty \text{ (bounded).}$$

Inserting Equation 3.16 into 3.15 and using the assumed kernel properties, leads to the following asymptotically unbiased expectation of $\hat{f}$

$$E\hat{f}(x,h) = f(x) + \frac{1}{2}h^2 f''(x) \int y^2 \mathcal{K}(y)dy + o(h^2).$$

This yields the bias expression

$$E\hat{f}(x,h) - f(x) = \frac{1}{2}h^2 f''(x) \int y^2 \mathcal{K}(y)dy + o(h^2). \tag{3.17}$$

Equation 3.12 for the variance can be rearranged in a similar way to

$$Var\{\hat{f}(x,h)\} = (nh)^{-1} \int \mathcal{K}(y)^2 f(x-hy)dy - n^{-1}\{E\hat{f}(x,h)\}^2. \tag{3.18}$$

Assuming that h is small and the number of observations n is large, $f(x-hy)$ can be approximated by $f(x) + o(1)$ and $\{E\hat{f}(x,h)\}^2$ by $\{f(x)+o(1)\}^2$. This leads to

$$Var\{\hat{f}(x,h)\} = (nh)^{-1} f(x) \int \mathcal{K}(y)^2 dy + o\{(nh^{-1}\}.$$

Adding the variance and the squared biased gives the MSE

$$MSE\{\hat{f}(x,h)\} = (nh)^{-1} f(x) \int \mathcal{K}(y)^2 dy + \frac{1}{4}h^4 f''(x)^2 \left(\int y^2 \mathcal{K}(y)dy\right)^2 + o\{(nh)^{-1} + h^4\}.$$

If we now integrate this term over x (assuming f integrates to 1), we obtain the *asymptotic* MISE (AMISE) as a simpler approximation to the MISE in 3.14

$$MISE\{\hat{f}(\cdot,h)\} = AMISE\{\hat{f}(\cdot,h)\} + o\{(nh)^{-1} + h^4\} \tag{3.19}$$

where

$$AMISE\{\hat{f}(\cdot,h)\} = (nh)^{-1} \int \mathcal{K}(y)^2 dy + \frac{1}{4}h^4 \int f''(x)^2 dx \left(\int y^2 \mathcal{K}(y)dy\right)^2. \tag{3.20}$$

3.6.1 Optimal Bandwidth

In order to find the optimal bandwidth in terms of the AMISE definition, we need to minimize Equation 3.20 by differentiating with respect to h and setting the result to zero. Then, solving for h yields

$$h_{AMISE} = \left(\frac{R(\mathcal{K})}{n \cdot \mu_2(\mathcal{K})^2 \int f''(y)^2 dy}\right)^{1/5}, \tag{3.21}$$

where $R(\mathcal{K}) = \int \mathcal{K}(x)^2 dx$ and $\mu_2(\mathcal{K}) = \int x^2 \mathcal{K}(x)dx$ are constants that only depend on the shape of the kernel $\mathcal{K}$. The formula 3.21 shows that h_{AMISE} is inversely proportional to the unknown quantity $\left(\int f''(y)^2 dy\right)^{1/5}$, which is a measure of the total curvature of f. This reveals that for densities with little curvature the bandwidth should be relatively large and vice versa. Moreover, formula 3.21 shows that the ideal bandwidth converges to zero as the sample size approaches infinity, but at a very slow rate of $\mathcal{O}\left(n^{-1/5}\right)$.

3.6.2 The Optimal Kernel

Observing formula 3.21 shows that besides the bandwidth depending on the unknown curvature f'' the AMISE error can also be reduced by an appropriate kernel function $\mathcal{K}$. If we insert the expression 3.21 for the ideal bandwidth h_{AMISE} into the formula 3.20 for the AMISE error, we can separate a constant factor $C(\mathcal{K})$ that only depends on the kernel shape

$$AMISE\{\hat{f}(.,h)\} = C(\mathcal{K}) \cdot \left\{ \left(\int f''(x)dx \right)^{1/5} \cdot \left(1 + \frac{1}{4n^{4/5}} \right) \right\}, \tag{3.22}$$

where

$$C(\mathcal{K}) = \mu_2(\mathcal{K})^{2/5} \cdot R(\mathcal{K})^{4/5}.$$

It can be shown that the constant $C(\mathcal{K})$ is minimized if

$$\mathcal{K}(t) = \begin{cases} \frac{3}{4\sqrt{5}} \left(1 - \frac{1}{5}t^2\right) & -\sqrt{5} \leq t \leq \sqrt{5} \\ 0 & \text{else.} \end{cases} \tag{3.23}$$

This kernel is called the *Epanechnikov* kernel since it was first suggested in density estimation by Epanechnikov (1969). It is optimal with respect to minimizing the AMISE and the efficiency of various kernels is often compared relative to the Epanechnikov kernel. However, from those efficiencies, one can observe that even for the simple *Box* kernel the efficiency is close to 1 (≈ 0.93) which shows that the kernel function is less significant in terms of error reduction. Therefore, the choice of the kernel is rather subject of other considerations such as the computational efficiency or the required degree of differentiability of the estimate. For instance, the Gaussian kernel (normal density) has derivatives of all orders but is one of the computationally most expensive kernel functions since it has unbounded support and involves evaluation of the exponential function. Figure 3.2 shows a density estimation sample for the most popular kernel functions in 1D.

3.6.3 Extension to Bivariate Density Estimation

The extension to two dimensions means that there are more degrees of freedom. First of all, a bivariate kernel has to be selected and secondly, one has to decide on the particular smoothing parametrization which is described by a symmetric 2×2 bandwidth matrix H. This allows for more flexibility and, if appropriately chosen, for faster convergence of the estimate $\hat{f}$. However, it also introduces more complexity into the estimator since more parameters need to be chosen. In case of a full bandwidth matrix, three parameters have to be selected, controlling not only the amount of smoothing (i.e. bandwidth in x and y) but also the direction (i.e. orientation of the kernel).

3.6.4 Bivariate Kernel Functions

Since we will deal with bivariate distributions throughout the thesis, we will describe several kernel functions in 2D and investigate what effect and which properties has the shape of the kernel function $\mathcal{K}$. As for 1D kernels $\mathcal{K}$ is often chosen to be a symmetric, unimodal density function. However, in 2D, the support of the kernel can have various shapes since we have more degrees of freedom. It might be a uniform disc or an oriented scaled ellipsoid for example. Therefore, it is common to use a symmetric 2×2 bandwidth matrix which is often simplified

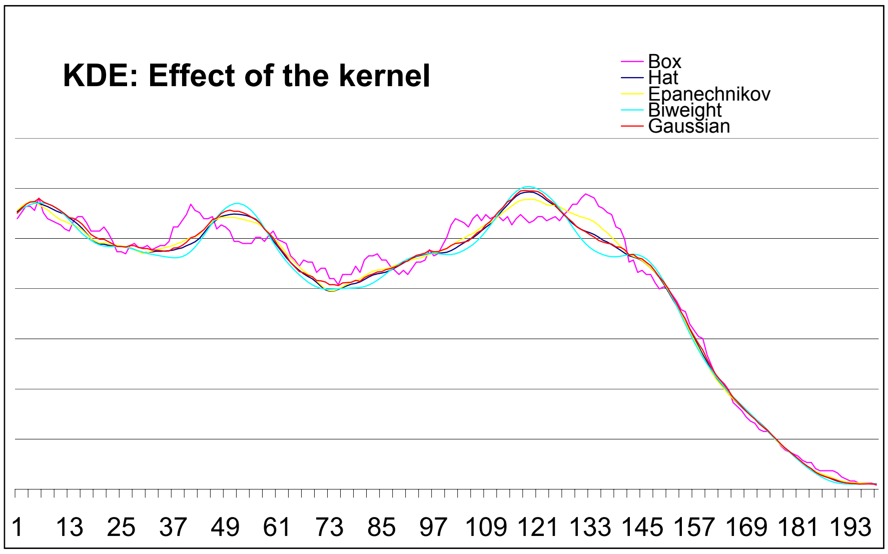


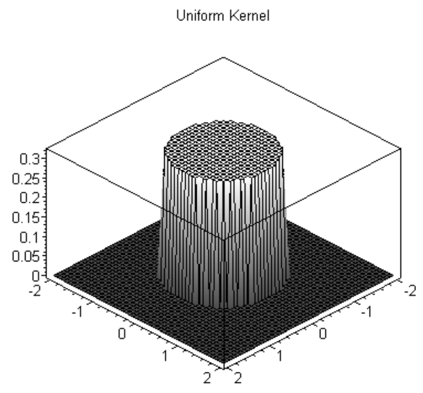
Figure 3.2: *Results of bivariate density estimation for 200 query locations along a line in 2D using 5 different kernel functions. It clearly shows that for all kernels except for the Box function, the estimates are very similar to the optimal* Epanechnikov *kernel!*

to be a diagonal matrix whose diagonal contains the bandwidth vector. However, we will only regard equally scaled kernels (disc support) with normalized bandwidth $h \leq 1$ (i.e. area $= \pi$). One is not only restricted to positive kernels but can also use kernels with negative tails such as the *Mitchell* kernel [55] or the classical *Sinc* kernel, which has an optimal representation in Fourier space (finite box). These kernels preserve or enhance edges and discontinuities. To our experience, besides being more expensive to evaluate, they also enhance the low frequency noise in the density estimates. Therefore, we will not regard such kernels.

3.6.4.1 Uniform (Cylindrical) Kernel

The simplest kernel is the uniform kernel, which has the shape of a cylinder in 2D. Because in 1D it is represented by a box, it is often referred to as *box filter*. It is defined by

$$K_b(t) = \begin{cases} \frac{1}{\pi} & |t| < 1 \\ 0 & else. \end{cases}$$

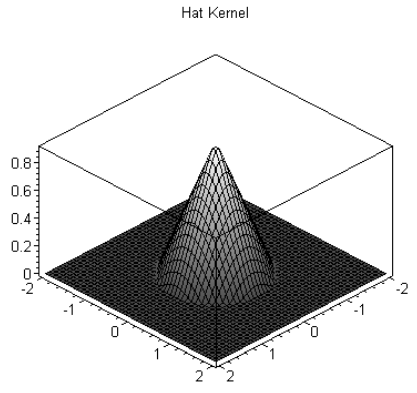


The uniform kernel has the least variance among all kernel functions and allows for fastest evaluation. However, the density estimation results exhibit staircase artifacts and do not have continuous first derivatives.

3.6.4.2 Hat (Cone) Kernel

The linear *hat* (or cone) kernel is simple but effective kernel and is often used in computer graphics. The efficiency with respect to the optimal kernel is ≈ 0.986. However, to calculate the distance t, one normally needs to compute a square root.

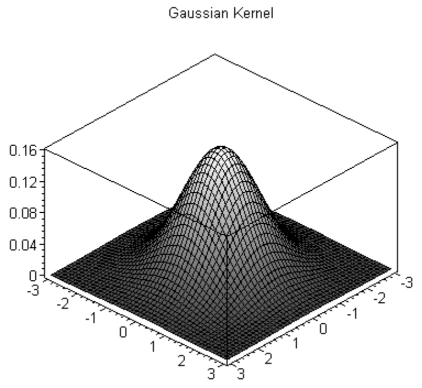
$$K_t(t) = \begin{cases} \frac{3}{\pi} \cdot (1-t) & |t| < 1 \\ 0 & else. \end{cases} \tag{3.24}$$



3.6.4.3 Gaussian Kernel

The *Gaussian kernel* (normal density) is the "queen" of all densities. It has derivatives of all orders and stays a Gaussian under Fourier transformation and convolution. However, it is expensive to evaluate and has infinite support.

$$K_g(t) = \frac{1}{2\pi} \exp^{-\frac{1}{2}t^2} \tag{3.25}$$

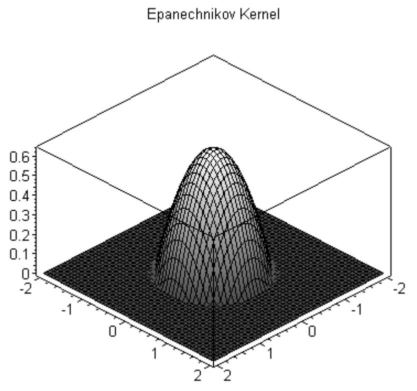


3.6.4.4 Epanechnikov

The *Epanechnikov kernel* is the optimal kernel with respect to minimizing the AMISE. It was given for the univariate case in formula 3.23. Here is the 2-dimensional version

The evaluation of the Epanechnikov kernel is very efficient since we only need the squared distance t^2. Hence, an expensive square root computation to obtain t as it is necessary for the triangular kernel is avoided.

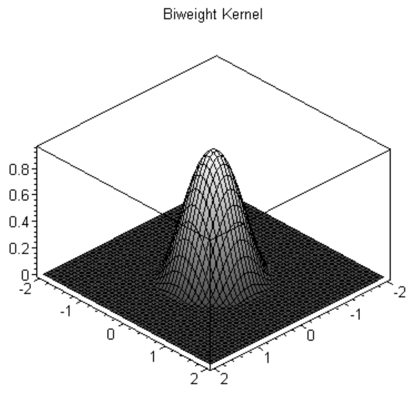
$$K_e(t) = \begin{cases} \frac{2}{\pi} \cdot (1 - t^2) & |t| < 1 \\ 0 & else. \end{cases} \tag{3.26}$$



3.6.4.5 Biweight Kernel

The *Biweight* (also called Quartic or Bisquare) kernel is closely related to the Epanechnikov kernel and has almost the same efficiency (≈ 0.994). However, it owns higher order smoothness and hence the resulting density estimates have higher differentiability properties. Its bell-like shape is somewhat similar to the normal density (Gaussian kernel) but the kernel is more efficiently calculated.

$$K_b(t) = \begin{cases} \frac{3}{\pi} \cdot (1 - t^2)^2 & |t| < 1 \\ 0 & else. \end{cases} \tag{3.27}$$



3.7 Bias Reduction Techniques

How should the bandwidth be selected in practice? This is a difficult problem since it depends on the unknown density function f that we are trying to estimate and whose second derivative can take any value. Therefore, much research work has been dedicated to this subject. We can see that Formula 3.21 also depends on the given parameter n and the values $\mu_2(\mathcal{K})^2$ and $R(\mathcal{K})$. Both values depend only on the kernel $\mathcal{K}$ and can be solved analytically.

3.7.1 Rule-of-Thumb

One way of computing the optimal bandwidth h is by assuming a simple known density, which often happens to be the normal density $N(0, \hat{\sigma}^2)$ where $\hat{\sigma}^2$ is an estimate of the variance σ^2. This is known as the "*Rule-of-Thumb*" proposed by Silverman[74]. Hence, h only depends on n, $\mathcal{K}$ and an estimate of the standard deviation σ. Using this simple parametric rule, the choice for h becomes

$$h = C(\mathcal{K}) \cdot \sigma n^{-1/5},$$

where $C(\mathcal{K})$ is a characteristic constant analytically solved for each kernel. For example:

Gaussian:	$C(\mathcal{K}_g) = 1.06$
Epanechnikov:	$C(\mathcal{K}_e) = 2.34$
Biweight:	$C(\mathcal{K}_b) = 2.78$

Despite its simplicity, this method is inefficient if the density f is far from normal (e.g. multi-modal) and may lead to oversmoothing of relevant features.

3.7.2 Plug-in Methods

A more general approach to the problem of bandwidth choice is to estimate the curvature $\int f''(x)^2 dx$ in a first step and then plug this estimate into the formula 3.21. This is referred to as **plug-in** method. The difficulty arises from the initially chosen bandwidth for estimating $\int f''(x)^2 dx$ which is different from the optimal bandwidth for estimation of $f(x)$.

3.7.3 Cross-Validation

Another common technique for bandwidth selection is known as **cross-validation**. It uses the leave-one-out technique to estimate the MISE (or AMISE). *Least-squares cross-validation* [65] is one popular method based on cross-validation. We will not derive the complete formula here and give a reference to Silverman's book [74] instead. The basic idea about least-squares cross-validation is that we can split the integral of the integrated square error (ISE) into three components

$$ISE(\cdot, \hat{f}) = \int (\hat{f} - f)^2 = \int \hat{f}^2 - 2\int \hat{f} f + \int f^2. \tag{3.28}$$

Now, minimizing the ISE boils down to minimizing $\int \hat{f}^2 - 2\int \hat{f} f$ since the last term is independent from $\hat{f}$ and therefore regarded as constant. The middle term still contains the unknown f. We define $\hat{f}_{-i}$ to be the density estimate constructed from all data points except X_i

$$\hat{f}_{-i}(x) = (n-1)^{-1} \sum_{j \neq i}^{n} \mathcal{K}_h(x - X_j). \tag{3.29}$$

This is called the *leave-one-out* estimate. It can be shown that using the *leave-one-out* method, we obtain an unbiased estimator for $\int \hat{f} f$

$$E\left(n^{-1} \sum_i \hat{f}_{-i}(X_i)\right) = E\left(\int \hat{f}(x) f(x) dx\right).$$

Minimization with respect to h can now be done from the data points themselves regardless of f by minimizing the score function

$$M_0(h) = \int \hat{f}^2 - 2n^{-1} \sum_i \hat{f}_{-i}(X_i) \tag{3.30}$$

or using the computationally simpler expression derived from M_0

$$M_1(h) = n^{-2} \sum_i \sum_j \mathcal{K}_h^*(X_i - X_j) + 2n^{-1}\mathcal{K}_h(0). \tag{3.31}$$

where $\mathcal{K}_h^*$ is defined by $\mathcal{K}_h^*(t) = \mathcal{K}_h^2(t) - 2\mathcal{K}_h(t)$.

Least-squares cross-validation (LSCV) works fully automatically and is an unbiased estimator. The drawback is the expensive computation. We need to perform n^2 kernel evaluations for a set of bandwidth values (or matrices) in order to minimize M_1 over h. The advantage of LSCV is that the computation is only dependent on the number of data points and therefore not explicitly dependent on the dimension of the data. Furthermore, the relative rate of convergence of LSCV improves for higher dimension [23]. On the other hand, a problem with LSCV is that it suffers from discretization (i.e. misinterpretation of discretization artifacts as high frequencies) and may select too small bandwidths leading to strongly undersmoothed estimates. Therefore, other estimators also based on cross-validation have been proposed such as the *biased cross-validation* (BCV) [70, 66] which attempts to minimize the AMISE instead of MISE. However, although asymptotically unbiased, BCV leads to more biased estimates than LSCV. As a remedy, there is an estimator known as *smoothed cross-validation* [57] which can be considered as a trade-off between both methods but is more complex and difficult to implement [89].

3.7.4 Computational Problems and Hints

In order to reduce the operations in the KDE formula, one should move the factor $\pi \cdot h^d$ out of the sum and multiply it at the end. Additionally, for kernels such as *Gaussian*, *Epanechnikov*, and *biweight* we can use a slightly modified version of the kernel $\mathcal{K}(\|x - X\|^2, h^2)$ instead of $\mathcal{K}\left(\frac{\|x-X\|}{h}\right)$. For instance, the Epanechnikov kernel is then defined by

$$K_e(t_2, h_2) = \begin{cases} \frac{2}{\pi} \cdot \left(1 - \frac{t_2}{h_2}\right) & t_2 < h_2 \\ 0 & else. \end{cases}$$

This way we avoid one multiplication, and more important one square root in the distance computation between x and a data point X_i. In case of a constant bandwidth for all points X_i, we can even avoid the division by h_2 inside the sum if we multiply the kernel function $\mathcal{K}$ with h_2 and divide the result of the sum over all kernel evaluations by h_2 at the end. Similarly, we can reduce the number of low-level operations for the other kernels.

Another algorithmic optimization can be achieved by reversing the order of computation for kernel density estimation. Instead of searching for each query location all nearest data points within the maximum bandwidth and computing the sum of all kernel evaluations, we can search for each data point all query locations within the local bandwidth (i.e. gather radius) of the data point. This is algorithmically superior if we use a tree for searching and if we have more query locations than data points. Moreover, it reduces the search footprint since we only need to sum up points within the local bandwidth of the data and not within the maximum support of all data points. This reduces the amount of kernel evaluation for the adaptive kernel density estimation.

Chapter 4

Photon Mapping using Density Estimation

In the previous chapter we have introduced the basic principles of density estimation and we have discussed various methods for density estimation used in statistics. We have shown that density estimation is a non-trivial problem and still an active research area. Now, we will make a connection to computer graphics. In this chapter we will focus on photon mapping. We will introduce the basic concept in a chronological order and explain several optimizations that can be made for photon mapping. First, we will start with photon emission, continue with the photon storage and radiance estimation followed by some analysis about photon mapping bias and several remedies. Finally, we will talk about the visualization of the photon map accompanied with acceleration methods. In Chapter 6 and Chapter 7 we will introduce two different variants of density estimation for global illumination that improve or accelerate the algorithm.

4.1 Overview

For quite some time, full global illumination computation including caustics and diffuse interreflection has been considered as too lavish and time-consuming especially in the production rendering of animation. The unbiased algorithms such as *path tracing* [37] or *Metropolis light transport* [84] suffer from high frequency noise in the image and the results converge very slowly. Therefore, global illumination was either neglected or roughly approximated. For example, artists have simulated global illumination by manually placing point light sources throughout the scene[1]. Nowadays, the computational power has increased massively while the hardware cost has reduced. Hence, global illumination becomes more and more an issue even in animation rendering. Nevertheless, unbiased algorithms are too expensive and biased algorithms are preferred. There are several *radiosity* methods [11, 87, 21] that explicitly work over polygonal representations and require an adaptation of the geometry to the illumination. Moreover, radiosity methods are not general enough and do not handle caustics or glossy light transport. What was needed was a method that could handle various BRDFs, complex models consisting of different kind of geometry and all this in high quality with efficient computation. The probably most general technique fulfilling these requirements is *photon mapping* (PM) introduced by Henrik Wann Jensen [33] in 1996. PM computes irradiance via density estimation of photons.

[1]The subjective approach of manually placing point light sources throughout the scene can be understood as a manual variant of *instant radiosity* [42].

Irradiance computation was shown to be a density estimation problem [30] and has become very popular in computer graphics since the nineties because of its ability to efficiently render caustics. With photon mapping global illumination became feasible, and since more memory was available, one could also afford the storage of a high number of photons. PM is commonly used as an extension of *Monte Carlo* ray tracing to efficiently simulate global illumination. Although its analogy to quantum physics is striking and there have been attempts to physical based simulations, the name ”photon” has a slightly different interpretation in computer graphics. Just the fact that a photon carries radiant flux is equivalent to the physical photons. However, we will neglect most quantum effects such as diffraction and polarization of light that are derived from the wave properties of photons. Moreover the number of photons is immense. For instance, a small light bulb (100 Watt) emits approximately $2 \cdot 10^{20}$ to $3 \cdot 10^{20}$ photons per second. We can store in memory only in the order of a few million ($\approx 10^6 \ldots 10^8$) photons. The disadvantage of photon mapping is the same as for all density estimation problems. In order to diminish the error in the irradiance computation, PM needs a huge number of photons and a robust bandwidth selection.

4.2 Photon Tracing

Photon tracing is the process of emitting photons from the light sources and tracing them through the scene. The photon hit points form the photon map that can be used for irradiance computation in a second pass.

4.2.1 Emission

First of all photons must be emitted from a light source. There are different types of light sources: point lights, area lights, directional lights, or physical correct lights with arbitrary geometry and distribution (e.g. neon tubes, television). The power of a light source is distributed to a set of photons. Each photon gets the same power (though different spectra) which is proportional to the total number of emitted photons but not to the number of stored photons in the map. Another important concept is that the photon’s power is independent from the power of a light source. Instead the *number* of emitted photons from a light source is chosen to be proportional to the light source’s power. However, this is not obligatory and even contradictory in the case of importance sampling.

The simplest case of a light source though not physical plausible is the *point light*. For a point light each photon gets the same origin and has only two initial degrees of freedom (i.e. direction). A random direction is chosen via explicitly sampling a (hemi-)sphere or via rejection sampling. The second kind of lights are the area light sources, which are physical plausible. For an area light source in addition to the direction a random position on the light source (often a quad) is chosen and the outgoing direction is proportional to the cosine of the outgoing angle with the geometric normal. Area light sources can be simple quads or realistic 3-dimensional light bulbs for example.

It is important to note that not the total number of emitted photons from all light sources is user defined but the number of photon hits to be stored. The number of emitted photons and stored photons in the photon map (see Section 4.2.3.1) can differ significantly. For example, there might be only a few photons contributing to caustic light paths which may cause several millions of photons to be traced until the required number of caustic photons has been stored.

Moreover, the required photon density for caustics and thus the number of photons in the caustic density estimation needs to be higher than for the "all-light-transport" *global photons*. Therefore, more photons need to be emitted for caustics. To do so the user specifies either a certain total number of photons for each *photon map* N_{max} driven by the memory limits or a rendering accuracy and computation time. Then the photons are emitted in a few iterations (3 to 5) with a fraction of N_{max} photons in the first iteration (e.g. $M_1 = N_{max}/$ maximum photon path length). For each following iteration $i+1$ we compute the ratio N_i/M_i of stored photon hits N_i to emitted photons M_i from the current and all previous iterations. Hence, for the next iteration $(N_{max} - N_i) \cdot M_i/N_i$ photons are emitted. The procedure continues until $N_i/N_{max} > 0.95$. To obtain a required number of caustic photon hits, it can happen that a huge number of photons needs to be emitted. This is the case if the specular surfaces as seen by the light sources are relatively small with respect to the whole model. In order to avoid many useless ray intersection tests for the photon emission, projection maps [35] can be used to detect initial caustic contributing directions. This however works only for direct caustics generation $(L(S)+D)$ where specular surfaces are directly seen by the light source.

4.2.2 Scattering

Once a photon has been emitted from the light source, it is traced through the scene as for standard (bidirectional) path tracing. However, the difference is that the photon carries flux (i.e. energy per time) whereas rays gather radiance. When a photon arrives at a scattering point on its path, it is either *absorbed*, *reflected*, or *refracted*. This scattering event is decided probabilistically based on the surface material. The decision is made by *Russian roulette* based on the surfaces' material properties and the photon is scattered using BRDF importance sampling.

Russian roulette has the advantage that the mean power of a photon is not affected by the reflectivity of the surface material as for normal sampling. Instead the power is only divided by the probability density from BRDF importance sampling. Russian roulette reduces the number of "unimportant" photons with little contribution while still being unbiased. It does not induce a dependence on the depth of the photon path as the exponential growth of the number of scattered photons in normal sampling does. Instead, the number of stored photons decreases with the depth (i.e. for each bounce). On the other hand, Russian roulette increases variance in the photon density because unlikely sampled regions in the scene will get fewer but equally powered photons. Most analytical BRDFs consist of a specular and a diffuse term which allows us to further exploit Russian roulette to decide upon the type of a scattering event. Hence, given a uniform random number ξ and material reflection coefficients for diffuseness ρ_d, specularity ρ_s with $\rho_d + \rho_s \leq 1$, we decide if a photon is scattered diffusely, specularly, or if it is absorbed

$$\begin{array}{rcr} \xi \in [0, \rho_d] & \longrightarrow & \text{diffuse reflection} \\ \xi \in (\rho_d, \rho_s + \rho_d] & \longrightarrow & \text{specular/glossy reflection} \\ \xi \in (\rho_s + \rho_d, 1] & \longrightarrow & \text{absorption} \end{array} \tag{4.1}$$

In case of a reflection, one can use a little trick to reduce the dimension in the random number generator: if ξ is smaller than the (average) surface albedo $\sigma = \rho_s + \rho_d$, we can exploit ξ to generate a "new" random number by $\xi' = \xi/\sigma$. Since σ and ξ are uncorrelated, ξ' is also a uniform random number between 0 and 1.

Russian roulette can also be used for reflection/transmission distribution given the material's transparency $\rho_t < 1$. It is also simple to extend the selection scheme to handle photons with multiple color bands where we need to consider the mean reflectance over all channels. A photon is stored in the photon map when it hits a diffuse or moderately glossy surface and continues the random walk until it is either absorbed or has experienced a maximum number of bounces.

4.2.3 Photon Storing

As mentioned before, photons are stored on diffuse and moderately glossy surfaces but not on highly specular ones since the probability that a photon arrives from a contributing direction within the narrow lobe of the specular BRDF is very low and zero for perfect mirrors. Instead, specular surfaces are sampled in mirror direction via BRDF importance sampling. The difficulty in the classification of the surface material which determines the photon storage is normally addressed by simply thresholding the diffuseness and glossiness material coefficients of the surface hit. A surface is then chosen to be either diffuse, glossy, or perfectly specular.

All useful photon hits are stored in the *photon map* by writing a photon hit record to the linear array which is sorted afterwards. Note that storing the individual photon hits ”decouples” the illumination from the scene model. A photon can be stored several times along its path, however, the longer the path the higher the variance of a photon due to several BRDF scattering events (in case of glossy BRDFs). A photon hit record can look as follows:

```
struct Photon
{
    float position[3]; // position (12 Bytes)
    float power[3];    // power in RGB or XYZ (12 Bytes)
    char  phi, theta;  // compressed incident direction (2 Bytes)
    short flag;        // flags and offset for the kd-tree (2 Bytes)
};
```

A photon hit must store information of the hit point in 3D, and the flux it transports which can be compressed to 4 Bytes if Ward’s shared-exponent RGBE-format [91] is used. However, it is computationally more efficient to store the power uncompressed as one float for each band (e.g. RGB) since the overhead for compressing and decompressing the power between `float` and `char` can ruin the floating point pipeline of modern processors. It is also possible to store only the power for one wavelength and each photons gets assigned a single wavelength that could be encoded in `flag`. This approach would be physically correct but can cause uncorrelated noise for each wavelength. In addition, more photons would be needed to get the same accuracy in the color spectrum. Therefore, it is often preferable to use the artificial concept of ”colored” photons. The previously shown representation of the photon record uses three color bands and comprises 28 Bytes in total which is not desirable with respect to alignment in memory. A size of 32 Bytes is preferable. However, the photon power uses 4 floats in our implementation because it also encodes the type of the spectrum rather than just 3 fixed bands. Therefore, our photon structure occupies 32 Bytes. As proposed by Jensen [33], the incoming direction of the photon is compressed to 2 Bytes in spherical coordinates (ϕ is the longitudinal angle and θ is the azimuth angle measured from the normal) allowing for $256 \times 256 = 65536$ directions. For diffuse photon hits (Lambertian surfaces) the incoming direction is used to check whether a photon arrived at the front (i.e. $\vec{\omega}_{in} \cdot \vec{n} \leq 0$) of the surface and for glossy surface hits it is used for

BRDF evaluation. The compression to 2 Bytes seems to be sufficient for simple culling tests and even acceptable for BRDF evaluation of photons on glossy surfaces. To avoid the computation of cosine and sine for the decompression of the direction (θ, ϕ) to (x, y, z), four lookup tables for cosine and sine of theta and phi each of 256 values are precomputed.

The `flag` field has different meanings depending on the data structure being used. It either stores only the splitting plane dimension of a kd-tree node (3 dimension $\Rightarrow$ 2 bits) where 14 bits are wasted or it is used as a counter for the number of photons in the leaf in the spatial kd-tree. For *reverse photon mapping* described in Chapter 6 we also use it to store the bandwidth compressed to 1 Byte for the *adaptive kernel density estimation*.

4.2.3.1 Photon Map

Obviously, for efficient searching of the nearest neighbors during density estimation, we need to organize the array of photons in a more appropriate layout than an unsorted list. Several data structures have been proposed among which the kd-tree is the most promising one since it is both adaptive and efficient. Other tree data structures such as an octree or bsp-tree might also be possible but were not employed in this work. The construction of the kd-tree involves sorting of the photons and is of complexity $\mathcal{O}(n \log n)$. The time spent for the construction is often negligible compared to the rest of the computation and pays off during rendering. It was proved that the complexity for searching the k-nearest neighbors in a balanced kd-tree is $\mathcal{O}(k + \log n)$ [4].

An alternative to the kd-tree is a regular grid. A grid allows for highly efficient nearest neighbor queries in constant time and is also slightly faster to construct since fewer sort operations and no balancing needs to be performed. On the other hand, a grid is not adaptive and it is difficult to choose the optimal grid resolution. If the resolution is to fine a lot of memory is wasted since most of its voxels are empty and contain null pointers. If the resolution is to coarse, we need to search within a large number of photons in voxels with a high photon density. A remedy is the hybrid grid where we use a coarse resolution for the grid and store a pointer to a sub-data-structure in each voxel which can be a simple list, a tree, or a grid itself (recursive grid). The hybrid grid can be considered as a trade-off between efficiency and adaptation. Naturally, the search efficiency is improved due to the decrease of the depth of the sub-trees in the voxels compared to a single large tree. However, the spatial kd-tree proposed in this thesis performs very similar to the hybrid grid but is more flexible as we will see in the next chapter where we examine various search data structures.

4.3 The Radiance Estimate

Given the photon map from the first pass, we can compute different types of statistics of the illumination in our scene. In the following, we will demonstrate how the photon map can be used to estimate the irradiance at an arbitrary point in the scene and hence how we can obtain an estimate for the reflected radiance.

4.3.1 Density Estimation

We already know that the photon map represents incoming flux and that a photon transports a fraction of the light source power. Therefore, a photon hit with the model indicates direct or indirect illumination received from a certain light source. However, a single photon has very little impact on the illumination. What matters is the number of photon hits per surface area (i.e. the density of photons). This leads to the definition of irradiance. The first methods based on photon tracing used illumination maps where the photon hits were accumulated in bins corresponding to texels of a textures [30]. Later on, adaptive approaches where published using tessellated geometry [61]. These method can be seen as histogram based density estimation which is known to be inferior to kernel density estimation (KDE) because KDE operates on the individual elements and yields smooth, continuous results whereas histogram results are piecewise constant (or piecewise linear if linear interpolated). Moreover histogram approaches have the problem of being dependent on the scene model (e.g. there might be regions in the scene where there are fewer photon hits than bins). Hence, the error is not eliminated if we increase the number of photons to infinity as for photon mapping with adaptive bandwidth selection (e.g. kNN density estimation).

Besides better density estimation, keeping the individual photon hits has also the advantage that we are able to estimate the density not only on diffuse surfaces (i.e. constant BRDF) but also on glossy surfaces since we store the incoming direction of photons and can evaluate the BRDF.

4.3.1.1 Relation to the Rendering Equation

To compute the reflected radiance from the photon density we need to derive a formula from the original rendering equation

$$L_r(x,\vec{\omega}_o) = \int_\Omega f_r(x,\vec{\omega}_i,\vec{\omega}_o) L_i(x,\vec{\omega}_i)(\vec{n}_x \cdot \vec{\omega}_i) d\omega_i, \tag{4.2}$$

where L_r is the reflected radiance at x in direction ω_o which is computed as the integral of the incoming radiance L_i modulated by the BRDF f_r over the hemisphere Ω, as described in Section 2.2.2. However, the photon map provides information about the incoming flux (i.e. energy per time) and we need to modify the formula. Since radiance is defined as

$$L_i(x,\vec{\omega}_i) = \frac{d^2\Phi_i(x,\vec{\omega}_i)}{(\vec{n}_x \cdot \vec{\omega}_i) d\omega_i dA_i}, \tag{4.3}$$

we can rewrite the integral as

$$L_r(x,\vec{\omega}_o) = \int_\Omega f_r(x,\vec{\omega}_i,\vec{\omega}_o) \frac{d\Phi_i(x,\vec{\omega}_i)}{dA_i}. \tag{4.4}$$

What is shown on the right hand side of formula 4.4, is simply the definition of irradiance and that's what we can compute by density estimation using the photon map. The incoming flux Φ_i at x can be approximated by the photon map using the nearest photons within radius r in the neighborhood of x. Each photon p has power $\Delta\Phi_p(\vec{\omega_p})$ and we can approximate the integral by a finite sum

$$L_r(x,\vec{\omega}_o) \approx \sum_{p=1}^{n} f_r(x,\vec{\omega}_p,\vec{\omega}_o) \frac{\Delta\Phi_p(\vec{\omega}_p)}{\Delta A}. \tag{4.5}$$

This formula is more convenient with respect to computation since the rendering equation 4.2 boils down to a simple sum of photon flux modulated by the BRDF. The cosine term cancels out, which is intuitively clear since the density of photons is inherently determined by the orientation of the surface to the incoming photon direction due to the photon-ray shooting. For example, if the surface normal is perpendicular to the photon's direction, then obviously the probability that this photon hits the (piecewise planar) surface is zero. The constant area ΔA can be moved out of the sum. For exact reconstruction one needs infinitely many photons. Otherwise, we just get a noisy or blurred version of the original integral depending on the finite area ΔA which is normally chosen to be a disc (πr^2) of radius r. We also see that the density from the photon hits can be considered as the estimation of the unknown irradiance since the photon hits are distributed according to the irradiance function. However, the irradiance and hence the density function is complex and in general unknown which restricts the computation to non-parametric density estimation. Formula 4.5 clearly shows the relationship to density estimation if we consider a simple box filter ($1/\pi$) and discard the BRDF in the sum. The difference to statistical density estimation is that photons may have different contributions (i.e. flux) and that outliers among the photons, mostly resulting from importance sampling, can spoil the density estimation as they increase variance. This often leads to disturbing low frequency noise (i.e. blurry speckles). On the other hand, the original function is usually smooth and does not contain high frequencies which keeps the error introduced by convolution of the density function low and in addition makes the function suitable for interpolation (e.g. irradiance caching). Remember that the optimal bandwidth depends on the second derivative of the function, which, in most cases, is very low resulting in a large optimal bandwidth for r.

The difficulty with the photon map is that the photon hits are distributed in three dimension but the density estimation is two-dimensional since the irradiance is measured on a surface. However, in statistical bivariate (2D) density estimation the domain is usually assumed to be uniform and continuous. For a bivariate distribution this means, we can assume that all data points lie on an unbounded plane. This is not the case for photon mapping. The domain contains high frequencies and discontinuities and is difficult to separate. There are usually two ways to deal with that problem: one either has to discretize the domain into different photon maps according to surface and geometric normal or one must detect and exclude "wrong" photons from the estimate. The former policy is similar to radiosity approaches. The latter is more general and is inherently achieved by using better kernel functions as described in statistical density estimation Section 3.6.4. Using standard symmetric kernels will not solve the problem completely but can significantly reduce the bias since photon hits from "wrong" surfaces are normally farther away from the density estimation point and therefore less weighted by the kernel (unless a box kernel is used). Another attempt to partially solve the problem is to compute the convex hull of the neighboring photon hits. This does not help for the "wrong-photon-inclusion" problem but helps to deal with discontinuities at the boundaries.

4.3.2 Bias in Density Estimation

In Section 3.6 we have already emphasized the difficulty with density estimation and derived expressions for bias and variance that merely depend on the bandwidth and the number of data points. This however is not the only source of bias in the photon mapping. As we have mentioned in the previous section, the domain in photon mapping is highly complex and we have to deal with various problems which are listed below. An illustration can be seen in Figure 4.1. More details on bias in photon mapping consolidated with an experimental analysis for several test cases can be found in Section 7.4.1 and in the recent paper by Schregle [69].

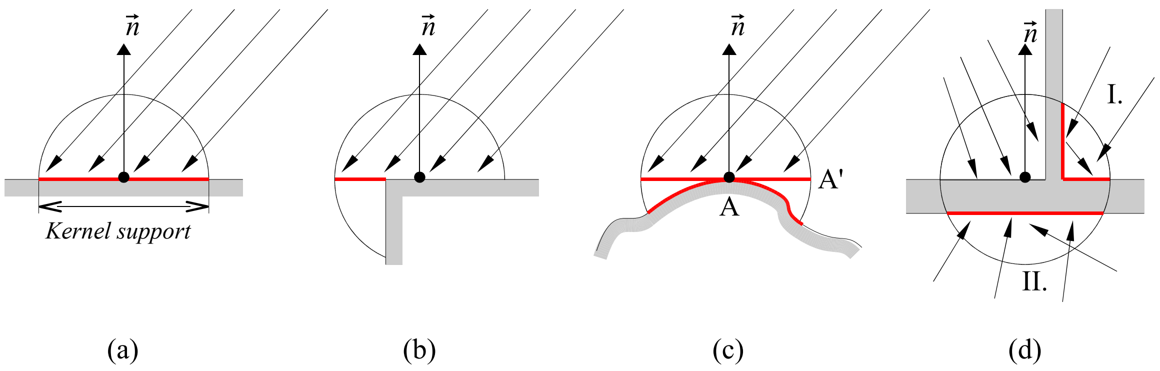


Figure 4.1: *Visualization of the bias sources in photon mapping: proximity bias (a), boundary bias (b), topological bias (c), and occlusion bias (d). The fat red line indicates the source of bias!*

4.3.2.1 Proximity Bias

Proximity bias is the most difficult form of bias in photon mapping and inherently exists in any density estimation method due to a convolution of the original irradiance function with the kernel function. It arises from including a finite number of photons in the vicinity of the actual query center (Figure 4.1.a). This leads to visible blurring of sharp details in the illumination such as caustics and shadow boundaries. Proximity bias cannot be avoided but reduced by increasing the number of samples (photons) and using adaptive bandwidth selection which however is often complex and computational expensive (e.g. cross-validation). However, bandwidth selection is a non-trivial problem and the number of samples is limited by the memory and processing time.

4.3.2.2 Boundary Bias

Boundary bias can be considered as not so severe as proximity bias though it is not trivial in the general case. Its source is the bounded domain which leads to underestimation of density due to overestimation of the surface area. The darkening on the visible surface boundaries is highly visible, hence final gathering is normally necessary. Boundary bias on flat convex surfaces can be handled via convex hull estimates. But the convex hull method fails on concave or curved surfaces because it assumes that the domain is planar and convex. Hence, we obtain a 2D projection of the convex hull. Other attempts to the boundary problem are the elongation of surface primitives, the mirroring of photon hits at the boundary. A more advanced possibility is the application of (non-symmetric) oriented kernel functions using 2x2 bandwidth matrices, which has been used in statistics. However, to our knowledge, there have been no attempts to apply such a method to photon mapping.

All these methods have drawbacks and are not robust enough for arbitrary topology and complex geometry. The most difficult type of boundary bias are high frequencies in the size of the geometry (tiny objects) which do not account enough photon hits to give a reasonable estimate. Even if we adapt the kernel support to the surface area as with convex hull estimate, in the limit, regarding infinitesimal small objects, we will not obtain any photon hits at all and therefore cannot tell anything about the density. Fortunately, there is a method that tackles the problem from another point of view. This method stores the photon paths rather than hits in a data structure called ray map and is therefore not bounded to surfaces as photon mapping is. The ray map proposal is part of this thesis and is described thoroughly in Chapter 7.

4.3.2.3 Topological Bias

Topological bias is related to boundary bias in the sense that the surface area is incorrectly approximated by a disc which assumes the surface to be planar in the neighborhood of the estimation query. This often leads to overestimation of photon density due to underestimation of the surface area on curved objects as shown in Figure 4.1.c. This form of bias is less problematic in the mesh-based approaches where photon hits are stored with the geometry since the density estimation area is computed from the primitives (e.g. triangles, quads) that build up the surface and can have any shape. Topological bias is partially solved using ray maps (Chapter 7).

4.3.2.4 Occlusion Bias

Occlusion bias can be seen as part of the topological bias and boundary bias. Thus, it is commonly not mentioned. However, we prefer to separate this kind of bias as it must be solved differently. The problem arises with including "wrong" photon hits that are not visible at the estimation point and are separated by thin objects such as walls. The bias becomes visible as "light leakage" through thin objects in particular for caustics. The example shown in Figure 4.1.d shows two classes (marked as I. and II.) of occlusion bias. Photons in group II. can be excluded by a simple heuristic that computes the dot-product of normal $\vec{n}$ with the incoming photon direction and discards all such photons that yield a positive dot product. This effectively culls all photons arriving at the back-side of the surface because they could never possible contribute if the surface is opaque. The second class of photons (I.) that are occluded by a thin wall is more difficult to detect since not just the azimuth angle (θ) with respect to the normal is decisive but also the longitudinal angle (ϕ) of the incoming photons. Trying to classify a single photon hit will not work because two photons hits, one on the "correct" side and the other on the "wrong" side, can obviously have the same incoming direction. In order to separate those groups, we need to consider the context of nearby photon hits and search for the discontinuity in the incoming photon direction and in the photon density.

Another possibility for detection of occlusion bias, is to test for occlusion via ray tracing [32]. This will remove *all* photons that are not visible to the density estimation point. However, there is also a drawback. What happens if photons are located on a curved surface? In this case there would be no visible photon at all (depending on the floating point data type precision). Therefore, one must either store the origin of the photons and compute the visibility via ray tracing from the origin to the density estimation point or one moves the photon hit point and the density estimation point a small distance back to where they came from (i.e. along their incoming direction) before testing the visibility. This way of occlusion testing is naturally more robust but also quite expensive and therefore, a better and simpler approach is to avoid occlusion bias already during scene modeling by increasing the thickness of walls.

4.4 Visualization

We have shown that the photon map can be used to approximate the local irradiance at a point in the model. As we have observed from the analysis of the photon map, the results are quite biased unless we emit a huge number of photons and use sophisticated bandwidth selectors. Therefore, one usually does not use the photon map directly for computing the visible irradiance from all contributing sources but merely splits the lighting integral of the rendering equation 2.2.2 into *four* disjoint light components [34], that are individually computed by either the photon map, by Monte Carlo sampling, or by a combination of both techniques.

Assume that we can separate the BRDF f_r into two terms. This is possible for most analytical BRDF models (e.g. Phong, Lafortune, Ward, Cook-Torrance) which consist of a simple diffuse component $f_{r,d}$ (normally Lambertian) and a (more complex) term $f_{r,s}$ for the specular lobe.

$$f_r(x, \vec{\omega}_i, \vec{\omega}_o) = f_{r,d}(x, \vec{\omega}_i, \vec{\omega}_o) + f_{r,s}(x, \vec{\omega}_i, \vec{\omega}_o). \tag{4.6}$$

Since light is additive, we can split the incoming radiance L_i into a sum of three components

$$L_i(x, \vec{\omega}_i) = L_{i,l}(x, \vec{\omega}_i) + L_{i,d}(x, \vec{\omega}_i) + L_{i,c}(x, \vec{\omega}_i), \tag{4.7}$$

where

- $L_{i,l}$ is the direct illumination from the light sources,
- $L_{i,d}$ is the indirect diffuse illumination from the light sources that has been scattered diffusely at least once,
- $L_{i,c}$ is the indirect specular illumination (caustics) from the light sources that has only been specularly reflected or transmitted.

Using the sum of BRDF components and the sum of incoming radiance contributions, we can reformulate the rendering equation by splitting the integral into:

$$\begin{aligned}
L_r(x, \vec{\omega}_o) &= \int_\Omega f_r(x, \vec{\omega}_i, \vec{\omega}_o) L_i(x, \vec{\omega}_i)(\vec{n_x} \cdot \vec{\omega}_i) d\vec{\omega}_i \\
&= \int_\Omega f_r(x, \vec{\omega}_i, \vec{\omega}_o) L_{i,l}(x, \vec{\omega}_i)(\vec{n}_x \cdot \vec{\omega}_i) d\vec{\omega}_i && (4.8) \\
&+ \int_\Omega f_{r,s}(x, \vec{\omega}_i, \vec{\omega}_o) \left(L_{i,c}(x, \vec{\omega}_i) + L_{i,d}(x, \vec{\omega}_i)\right) (\vec{n}_x \cdot \vec{\omega}_i) d\vec{\omega}_i && (4.9) \\
&+ \int_\Omega f_{r,d}(x, \vec{\omega}_i, \vec{\omega}_o) L_{i,c}(x, \vec{\omega}_i)(\vec{n}_x \cdot \vec{\omega}_i) d\vec{\omega}_i && (4.10) \\
&+ \int_\Omega f_{r,d}(x, \vec{\omega}_i, \vec{\omega}_o) L_{i,d}(x, \vec{\omega}_i)(\vec{n}_x \cdot \vec{\omega}_i) d\vec{\omega}_i. && (4.11) \\
& && (4.12)
\end{aligned}$$

The first integral in 4.8 is the direct illumination computation and is computed via (importance) sampling of the light sources. The second integral in 4.9 represents the specular reflection/refraction of light in direction to the observer (camera) and is computed via Monte Carlo ray-tracing. The third integral in 4.10 computes the diffuse reflection of all incoming specular light paths which are called *caustics*. Caustics are best solved by the photon map. And the last integral in 4.11 represents all indirect diffuse illumination that is diffusely reflected in direction of the observer.

The first two integrals are solved via Monte Carlo ray-tracing from the camera since they contain high frequencies in the illumination due to occlusion (such as shadow boundaries) and in the BRDF (specular reflection/refraction). The integral for caustics also contains higher frequencies because the light path consists of specular BRDFs. Nevertheless, the caustics integral *should* be solved with the photon map since all other approaches converge too slowly to a "noise-free" visualization. Since caustics are the only contribution that use the photon map directly to compute the visible radiance, it is very important to have enough caustic photons in the map. Therefore, one splits the photon map into two separate maps: one that only stores caustic path photon hits and a second that stores all sorts of photon hits. We will refer to this later.

The photon map is not only useful for estimating irradiance but also to direct the costly Monte Carlo sampling to *important* locations in the scene in order to obtain a faster convergence of the integral as for example in the direct illumination.

4.4.1 A Two-Pass Algorithm

The sum of integrals in 4.12 was not derived by chance but as a better starting point for using the photon map. The first two integrals are still computed via Monte Carlo ray tracing. However, the last two integrals of equation 4.12 are the most time-consuming ones if solved by pure Monte Carlo sampling. Therefore, using the photon map, we can rapidly accelerate the computation of caustics and indirect diffuse illumination.

Because caustics are estimated directly from the photon map, they need a higher density of photons. Therefore, one splits the photon map into two separate maps, one for caustic photons only and one for all photons.

Although for caustics, the "best" known way to compute the radiance contribution is by using the photon map directly, it is recommended to prepend an initial Monte Carlo sampling step for the diffuse indirect illumination computation. This step is called *final gathering* since we gather energy from the photon map.

4.4.2 Final Gathering

We have discussed that photon mapping exhibits low frequency noise and several forms of bias in the estimate. Although it is practically possible to reduce this error with several heuristics, there will always be a certain setting where such algorithms will fail. Therefore, in order to reduce the visible error, we take another way and seize the problem by "averaging" the error over many density estimation points in the scene. Since the error of density estimation is low in planar almost uniform regions, which is usually the largest part of the surface of most scenes, we reduce the overall error and balance it across all radiance computation points. To do so, we start with importance sampling the associated BRDF at the point visible to the camera and shoot several final gather rays (FGR) across the hemisphere. At each FGR hit point we compute the radiance estimated from the global photon map. Since Monte Carlo importance sampling is a random process, we will exchange the bias artifacts arrising from the photon map with random noise, which is less disturbing to the eye and vanishes with the number of samples.

Final gathering does not suffer from boundary bias and averages the error from density estimation between many radiance samples. However, the error does not vanish but stays mostly constant across the image. Final gathering can also easily handle complex BRDFs (e.g. glossy material) which is difficult (or impossible) using only density estimation from the photon map. The covered eye paths by final gathering are $E(S)*(D|G)(S)*(D|G)$ and the complete final gathering light paths are $L(D|G|S)*(D|G)(S)*(D|G)(S)*E$.

The difficulty using a two-pass Monte Carlo approach arises in the context of combining both methods: density estimation from the global photon map and final gathering. A common approach is to shoot a large number of final gather rays across the hemisphere whenever a ray on the eye path hits a diffuse surface. In most cases this ray is a primary ray and is directly seen by the eye. In order to suppress the noise to an imperceptible level, the number of final gather rays should to our experience be set from 400 up to 6000 samples per hemisphere. The number depends strongly on the frequency of the indirect light in the model which is unknown during rendering. High local densities in the photon distribution and indirect caustics often lead to poor estimates. To choose a (locally) sufficient number of final gather rays is still an open problem in photon mapping. Often it is addressed by simple assumptions or initial test renderings of the scene. A more advanced approach is to query the photon map at the final gather location to direct the final gather rays in important directions with radiance contribution [34]. These are sampled from a discrete cumulative distribution function which is constructed over the incoming photon directions.

For diffuse surfaces naive Monte Carlo sampling over the hemisphere is the standard approach. However, for glossy surfaces we need to take care when to combine the photon density estimation with final gathering. For example consider a highly glossy surface with a narrow specular lobe. If we shoot final gather rays via BRDF importance sampling, most of these final gather rays will eventually land in a small neighborhood in particular if the hit point is very close to the origin of the final gather. Thus, the radiance estimate from the photon map will be almost the same for all rays and the resulting radiance for the pixel will still contain the low frequency noise from the photon density estimation. As a simply solution we accumulate the diffuseness of the surface BRDFs (ρ_c) along the eye path and use the photon map only when a user-defined threshold (e.g. $\rho_c > 0.7$) is reached.

In order to render indirect caustic paths $L(D)+(S)+(D|G)(S)*E)$ it is important to note that final gather rays should be reflected/transmitted whenever they hit a specular surface. Indirect caustics can have an important contribution in reality. For example consider the indirect light coming through a window at noon when the sun is not directly visible through the window. This light is mostly diffusely reflected in the outdoor environment and illuminates the entire room.

4.4.2.1 Tricks and Hints

Here we present a few improvements for final gathering concerning the density estimation at the final gather ray hit point. First of all, for final gathering a smaller bandwidth (gather radius) should be used than for the direct visualization of the photon map because we do not see the low frequency noise from single final gather rays but we do see the light leakage in corners if the radius is too large. To our experience 30 to 100 nearest neighbors are sufficient. Actually, the number of nearest neighbors (i.e. the bandwidth of a final gather ray) should depend on four criteria:

1. the total number of photons n_p,
2. the fluctuations in the density of photons (rather than the density itself!),
3. the number of final gather rays per pixel,
4. and the distance the FGR traveled.

The first and second criteria are obvious since the number of samples and the second derivative of the density are included in the formula for the optimal bandwidth with respect to the AMISE minimization (Section 3.6.1). Corresponding to the AMISE the bandwidth should decrease with a very slow rate proportional to $1/\sqrt[d+4]{n_p}$, where d is the dimension, hence, in our case (for $d = 2$) with $1/\sqrt[6]{n_p}$. The number of nearest neighbors (k) is then proportional to $(1/\sqrt[6]{n_p})^2 = 1/\sqrt[3]{n_p}$ since k is proportional to the gather area (i.e. squared bandwidth). Hence, we propose to use $k = \sqrt[3]{n_p}$, which yields about 30 to 100 nearest neighbors for about 30,000 to 1,000,000 photons. This has also worked well according to our practical observation.

The second point is the fundamental problem of density estimation and should not be addressed by final gathering because it is *the reason* why we need to apply final gathering before the actual density estimation. Furthermore, it is already roughly approximated by the kNN technique.

The third point says that the gather radius (or the number of nearest neighbors) should decrease with the number of final gather rays N. Otherwise, we will hardly reduce the bias (due to a too large bandwidth) from density estimation but only reduce the noise. It is well-known for Monte Carlo integration that the error σ is proportional to $1/\sqrt{N}$. Hence, more final gather rays decrease the variance (i.e. noise) with rate $\sigma^2 \propto 1/N$. On the other hand, a smaller bandwidth R increases the variance in the bi-variate (2D) density estimate with $1/R^2$ [74]. This means if we want to reduce the variance by a factor of 2, we can either double the number of final gather rays or increase the gather radius by a factor $\sqrt{2}$, hence double the gather area. The relation between the number of final gather rays and the gather radius is not trivial and we do not explicitly induce a dependence in the implementation but leave it to the user who should appropriately set the number of nearest neighbors to be used for final gathering. Moreover, according to our experience, the performance of final gathering depends strongly ($\approx$ linearly) on the average k nearest neighbor photons found per final gather ray. If it is set too high, cache misses are more likely to occur which will decrease the performance.

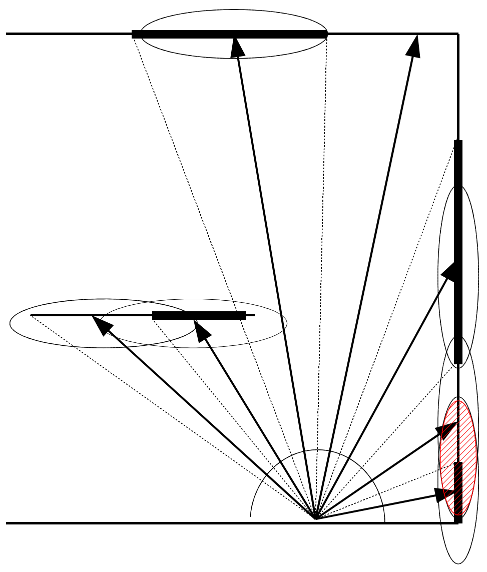


Figure 4.2: *The area covered by stratas (black thick lines) for final gathering of diffuse indirect illumination (in 1D) depends on the distance and orientation of the surface hit by a FGR. Note how the density estimation area of nearby FGR hits overlap yielding a similar (biased) radiance estimate.*

How the traversal distance of a FGR depends on R is difficult to state. But it is intuitively clear that close surfaces should be assigned a smaller R since the density of FGR hits decreases quadratically with the distance of the surface (assuming same orientation). Keeping a constant R (or kNN number) for all FGRs yields in a significant fraction of FGR hits on close surfaces using almost the same set of photons since the distance between the FGR hits is much smaller than R (see Figure 4.2). Consequently, neighboring FGRs will get a similar (biased) radiance estimates. Therefore, the error is much larger in corners or nearby surfaces than for open areas. (This is also the reason why irradiance caching uses the harmonic mean distance for spacing the irradiance cache locations.) We tackle the problem of final gathering from close surfaces (e.g. corners) by utilizing a secondary final gathering pass. If the hit distance of a primary FGR is below some scene size dependent threshold, we initiate a secondary final gather with fewer rays. The number of secondary FGRs (N_{2nd}) should decrease exponentially with the depth (D) of the final gather. A depth of 2 is sufficient since everything beyond this level would be too expensive to compute. In our implementation we set $N_{2nd} = N_{1st}/2^D$, where N_{1st} is the number of primary final gather rays. Additionally, N_{2nd} is bound to have at least 20 and at most 50 secondary final gather rays.

Since the number of kernel evaluations is very high even for normal rendering settings (see Section 6.6.1), it is recommended to use only simple kernels for density estimation, for example the uniform kernel (Section 3.6.4.1) or the Epanechnikov kernel (Section 3.6.4.4).

4.4.2.2 Choosing the Gather Radius

At each final gather ray hit point the neighboring photons have to be searched in order to compute the irradiance by density estimation. This is usually the most expensive procedure during final gathering. The search domain is often constrained to a sphere (a box is also possible) and the gather radius corresponds to the bandwidth in density estimation. For the classic k-nearest neighbors density estimation, the bandwidth is automatically determined by the local density of photons. However, searching naively the entire domain for the k-nearest photons is an overkill. Therefore, an upper bound for the initial radius is chosen in which we search for the k-nearest photons. The choice for the initial radius strongly influences the search performance. Often a constant radius is selected that is either set manually by the user or computed heuristically from the average photon density in the whole scene [9]. We propose to use an *adaptive* initial radius derived from a precomputed estimate of the local density in a spatial kd-tree. Such a spatial kd-tee is described in Section 5.2.1. We compute a pessimistic estimate of the density in each leaf/sub-tree of the kd-tree using the diagonal of the associated bounding box and the number of photons in the sub-tree's bounding box. This approach is described in Section 6.5.2. Additionally, the resulting gather radius is clipped by a maximum bound which is set manually by the user but could be derived heuristically from the size of the scene. We obtain a tight gather radius and the expensive kNN search is accelerated. The final gathering process is modified as follows: for each FGR we search for the leaf in the kd-tree where the FGR hit point is located in and return the initial gather radius stored in the leaf. This is done in $\mathcal{O}(\log(n/m))$, where m is the number of photons per leaf. In the next step we gather the k-nearest neighbor photons starting with the initial radius. The achieved speedup for various scenes is between 10% and 25% compared to a constant initial radius and includes the search for the tree leaf to obtain the initial radius. In both cases the maximum bound for the initial radius is set to the same value (5% of the scene diameter) for fair comparison and the resulting number of found nearest neighbors per FGR is the same. For the k-nearest neighbors search we use a fast heap implementation to quickly remove the farthest photon (i.e. $(k+1)$-th photon) from the center of the query.

4.4.3 Direct Illumination

The direct illumination integral is usually computed analytically given the position and power of all light sources and their visibility is computed by Monte Carlo sampling. It would not be sensible to apply importance sampling of the BRDF to compute the direct illumination since the probability of a ray, chosen randomly across the hemisphere, to hit a small light source would be very small and zero in case of a point light. Therefore, the direct illumination is computed by sampling the area of all light sources according to their power. This is the common way if we are dealing with a couple of small light sources. However, for large complex scenes with many light sources, sampling all of them one by one would be too exhaustive since most of them are perhaps not contributing (e.g. many rooms). Fortunately, the photon map can help in the evaluation of the "importance" of a light source since it can provide us information about the contribution of all light sources to a certain location in the scene. This can be done by searching the nearest hits from all "first-bounce" photons in the neighborhood of the point of interest [43]. If we know from which light sources the photons came from, we can quickly determine the energy contribution of every light source and eliminate the ones that are occluded. It requires only a small extension to the standard photon mapping algorithm: the photon structure must include an index to the light source and a flag to indicate if it is a direct photon hit (first bounce). Alternatively, direct photon hits could also be stored in a separate photon map.

4.5 Acceleration and Approximation Methods

This section shows a few important techniques for making photon mapping implementations more efficient. In general, one can classify these methods into interpolating algorithms such as irradiance caching [95], piecewise approximating algorithms (e.g. irradiance precomputation [8, 10], and simple heuristics. Interpolation is used in global illumination where there is an assumption of smoothness in the radiometric quantity. For instance, all radiosity methods use interpolation in the form of surface discretization. They adapt to the irradiance smoothness by adaptive geometric subdivision, e.g. [25]. Ward et al. [95] propose irradiance caching as a means of computing indirect diffuse inter-reflections in a distribution ray tracer [93]. They exploit the smoothness of the indirect illumination by sampling the irradiance sparsely over surfaces, caching the results and interpolating them. For each ray hitting a surface, the irradiance cache is queried. If one or more irradiance records are available, the irradiance is interpolated. Otherwise a new irradiance record is computed by sampling the hemisphere and added to the cache. In this way, the cache gets filled lazily, progressively in a view dependent manner. As it gets filled, more and more irradiance computations can be carried out by interpolation. Ward uses an octree for storing the irradiance records. In [94] the interpolation quality is improved by the use of irradiance gradients. Recently, the irradiance caching algorithm has been partially re-implemented on graphics hardware [19] resulting in a speedup of about two orders of magnitude compared with the Radiance rendering system [93].

4.5.1 Irradiance Caching

Irradiance Caching is a widely used method to reduce the amount of expensive final gathering for the diffuse lighting computation. It builds on the observation that diffuse indirect illumination changes usually slowly across surfaces. Hence, it is a good candidate for interpolation. Another observation is that illumination changes faster in regions with nearby objects yielding in color bleeding or shadowing due to occlusion. Putting these together, one can build a formula that estimates whether a point in the scene visible through a pixel should be re-computed via final gathering or interpolated by surrounding samples while keeping the error on constant level. The difficulty in this approach is how to efficiently find the surrounding samples for interpolation and what formula and parameters to choose for the irradiance estimation. Several methods have been proposed mostly based on some heuristics. One of the most commonly used models is the model of the split sphere [95]. It assumes a hemisphere above point x_i that is black on one side and white on the other representing the worst case change in the irradiance as we move away from x_i. There are two possible causes: a change in surface location and a change in surface orientation (i.e. normal variation). From this principle one can derive the following error estimation [95]:

$$\epsilon_i(x, \vec{n}) = \left\{ \frac{4}{\pi} \cdot \frac{\|x - x_i\|}{R_i} + \sqrt{2 - 2\vec{n} \cdot \vec{n_i}} \right\}, \tag{4.13}$$

where x is the current surface location, $\vec{n_i}$ is the normal at position x_i. R_i is the harmonic mean distance to the objects seen from x_i. It is defined as:

$$R_i = \frac{M \cdot N}{\sum_{\theta}^{M} \sum_{\phi}^{N} \frac{1}{\|RT(x_i,\theta,\phi) - x_i\|}}, \tag{4.14}$$

where $RT(x_i, \theta, \phi)$ is the first hit location for the final gather ray shot from x_i through the hemisphere strata defined by θ, ϕ (in the sense of stratified sampling). The weight for the contribution of x_i to x is inversely proportional to the error computed by equation 4.13:

$$w_i(x, \vec{n}) = \frac{1}{\epsilon_i(x, \vec{n})} \tag{4.15}$$

The value of this weight tell us if the irradiance at point x_i can be re-used for interpolation at point x. The higher the weight the better the estimate. The user provides an upper threshold for the error (referred to as a) and the weight has to be greater than its reciprocal:

$$w_i(x, \vec{n}) > \frac{1}{a} \tag{4.16}$$

Finally, to compute an estimate of the irradiance at x, we compute the weighted average of all previously computed irradiance values whose weight is greater than $1/a$:

$$E(x, \vec{n}) \approx \hat{E}(x, \vec{n}) = \frac{\sum_{i, w_i > 1/a} w_i(x, \vec{n}) E_i(x_i)}{\sum_{i, w_i > 1/a} w_i(x, \vec{n})}. \tag{4.17}$$

If there is no previously computed irradiance value with sufficient weight, we compute a new one (by final gathering the hemisphere and querying the photon map at the hit points). In our implementation we use a slightly different approach for the decision whether to recompute or interpolate. Originally, only one cache entry with a weight greater than $1/a$ is sufficient for interpolation. This however results in visible artifacts. Therefore, instead of only considering the sum of weights, we also account for the density of cache entries and penalize single cache entries. We decide to interpolate only if

1. there are at least k (e.g. $k = 4$) reusable neighboring cache entries (with $w_i > 1/a$),
2. or the sum of weights is greater than $2k/a$.

The first criteria ensures that we use in most cases k computed irradiance values for the interpolation, where k should be at least 2. (Otherwise we would get a piecewise constant approximation.) The second criteria avoids clustering of cache locations in a small area because it can happen that we place k cache entries next to each other until we have k entries to continue with interpolation. Intuitively, this means that a cache entry is reusable if its weight is high enough to compensate the k nearest entries (i.e. if we are very close to the entry). The factor $2k$ is just a heuristic based on the observation of the sample placement and could be chosen differently.

Computing the weights for all irradiance values at each sampled point visible to the camera, is very costly. It is clear that each irradiance value is only useful in a small neighborhood of a sample point x. Hence, we can limit the search to a region where the error with respect to the distance is below the user define threshold a. This maximum distance $R_{i,max}$ can be derived from Equation 4.13 if we ignore the surface orientation (i.e. the normals are assumed to be equal):

$$\|x_i - x\| < a \cdot R_i = R_{i,max} \tag{4.18}$$

$R_{i,max}$ only depends on the user defined threshold a and its harmonic mean distance R_i. This is intuitively clear since the irradiance changes faster in regions that are close to other surfaces. The result is visible as color bleeding or shadowing due to occlusion. Therefore, the harmonic mean distance is a good indicator to irradiance gradients.

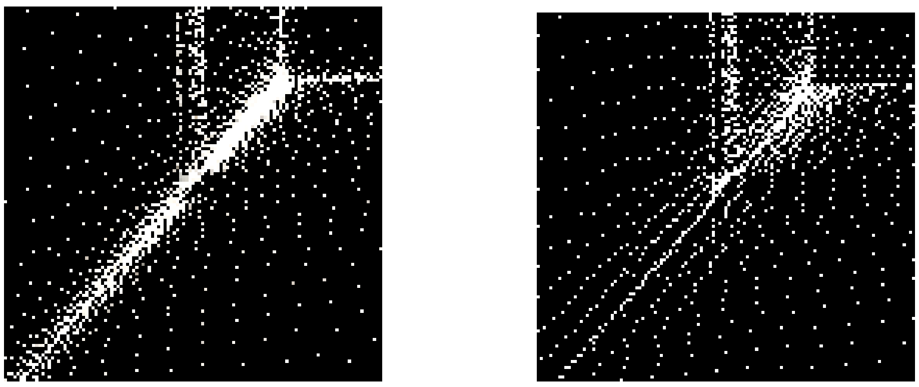


Figure 4.3: *Results without (left) and with bounding (right) the maximum radius R_i for the influence of an irradiance cache entry.*

The original proposed formula 4.13 used for computing the weights places too many samples near corners or edges and too few in open planar regions. Thus, it is a common procedure to set lower and upper bounds for the harmonic mean distance of a sample. The lower bounds ensures that less samples are computed in corners and the upper bound generates more samples in planar regions (see Figure 4.3).

In order to search efficiently for cached irradiance values, one usually uses an octree structure. A voxel is split into eight sub-voxels if the overlapping bounding box of the inserted sample is smaller than the voxel (i.e. has a shorter diagonal). See Figure 4.4 for a simplified example in 2D using a quad-tree instead. Therefore, when we are searching for re-usable cache entries (with sufficient weight), we first query the octree by recursively tracing down the sub-tree voxels

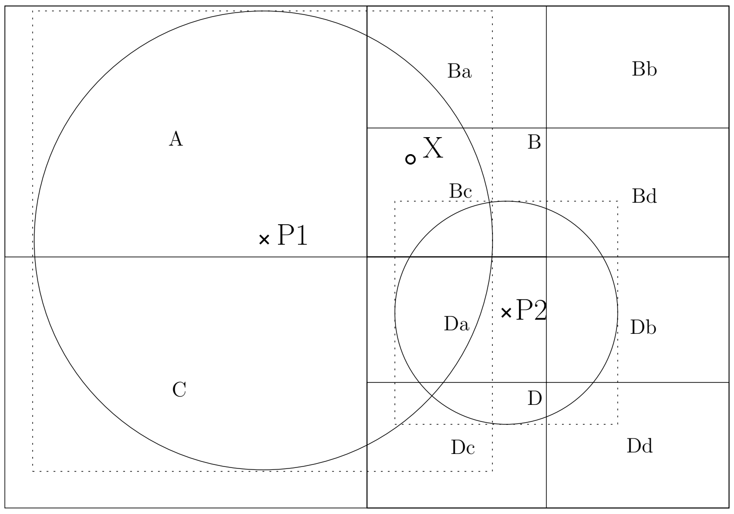


Figure 4.4: *Example of inserting two cache samples into the quad-tree. Sample P_1 is inserted into quads A, B, C, D whereas P_2 is inserted one depth level below P_1 into the quads Bc, Bd, Da, Db, Dc, Dd.*

that contain the current query location. Only the weights of the cached irradiance values in those voxels are computed and summed at the end. The sum of those weights determines if the current location can be interpolated. In Figure 4.4 a query at location X will search for cache entries in voxel B and its sub-voxel Bb computing the weights for P_1 and P_2. However, only P_2 may be re-used. P_1 can be rejected early.

In practice, one does not only consider the computed weight of a particular cache entry but also adds three additional heuristics: Skip cache entry i if:

1. surface orientations of x_i and x deviate more than a given constant threshold (see Figure 4.5.a), i.e. $(\vec{n} \cdot \vec{n}_i) < \cos(\alpha_{max})$, $\alpha_{max} \approx 10°$,

2. the current query location x is below the tangent plane of the cache entry x_i in query or vice versa (see Figure 4.5.b), i.e. $|(x_i - x) \cdot \vec{v}_{avg}| > \cos(\beta_{min})$, where $\vec{v}_{avg} = \vec{n}_i + \vec{n}$, and $\beta_{min} \approx 85°$,

3. distance between x_i and x in screen space is greater than a constant threshold pd_{max}, i.e. pd_{max} represents maximum distance between pixel coordinates.

The third condition ensures that cache entries that are close enough in object space but large in screen space (i.e. close to the camera) are not re-used. In our implementation, we also added a constant threshold to limit the minimum distance in screen space (about half a pixel) between samples that are neighboring which usually yields in fewer computed samples at the horizon far away from the camera. Table 4.1 shows the statistics for the rendering of the sponza scene and the icido scene with irradiance caching. Note that the percentage due to distance clipping is an early rejection due to the maximum possible distance in the split-sphere error computation. For the rendering 700.000 photons were used. One thousand final gather rays were traced for the sponza scene at each irradiance cache location and two thousand final gather rays were traced for the icido scene. The irradiance cache introduces little error when rendering the sponza scene since this scene is completely diffuse and contains low frequency illumination. Even with a large error the results look satisfactory. However, despite its general usage, irradiance caching can still lead to artifacts in case of higher frequencies in the indirect illumination and does not handle

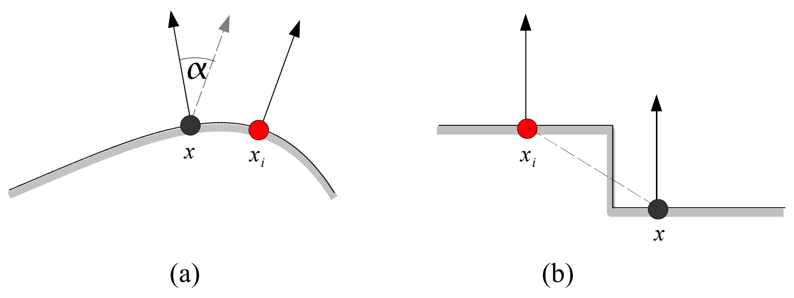


Figure 4.5: *Early rejection due to normal deviation (a) of cache entry (x_i) and early rejection due to vertical deviations (b) (stairs effect).*

	Sponza	Icido
FGRs	16, 148, 000	48, 431, 970
Total Queries	1, 535, 998	919, 186
Successful Queries	1, 519, 850 (98.95%)	894, 546 (97.32%)
Error Threshold	1.1	0.12
Octree Depth	7	7
Average Neighbors	4	4
Rejected Samples	136, 618, 199	48, 003, 887
R_{dist}	73.99%	71.09%
R_{normal}	14.59%	9.12%
R_{pixel}	6.97%	8.61%
R_{error}	2.97%	1.52%
R_{stairs}	1.58%	9.66%

Table 4.1: *Irradiance cache statistics for two scenes with different complexity: the sponza scene rendered with 1000 times 5 final gather rays per pixel in a resolution of 640×480 pixels and 700.000 photons, and the icido scene rendered with 2000 times 3 final gather rays per pixel with same resolution and same number of photons. The rows from top to bottom: the total number of traced final gather rays, the number of queries to the cache ($\approx$ the number of primary rays), the number of interpolated irradiance points, the maximum allowed error (a), the average depth of a cache entry in the octree, the average number of neighbors used for interpolation of the irradiance, the total number of rejected cache entries during the search of reusable neighbors, the percentage of such rejections due to distance clipping, normal deviations (Figure 4.5.a), screen space distance clipping, split sphere error (a), and stairs effect (Figure 4.5.b). Note that R_{dist} is also included in R_{error} and is tested before to avoid the expensive computation of R_{error}.*

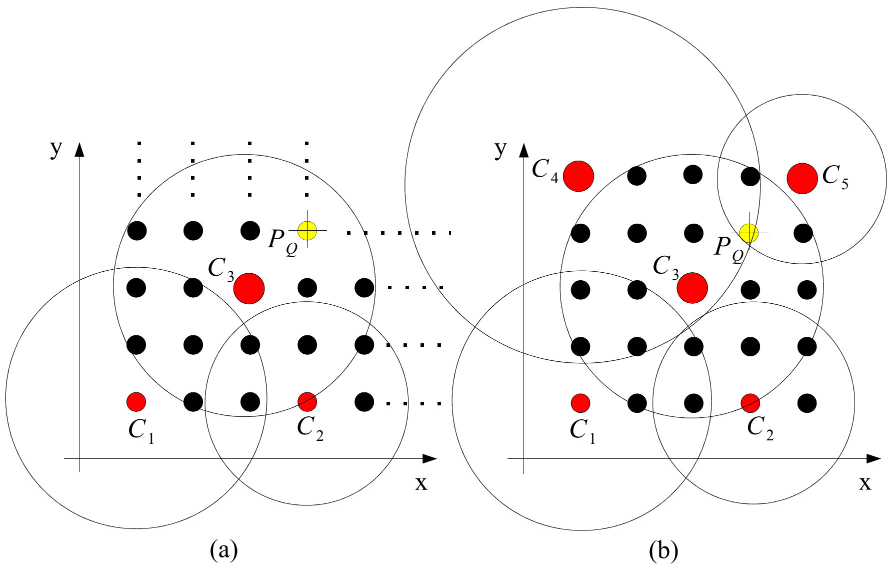


Figure 4.6: *Irradiance cache interpolation at point P_Q using naive line by line frame sampling (a) finds only the valid cache sample C_3 and misses potential samples C_4 and C_5. Using the final pass (b) all valid cache samples (C_3, C_4, C_5) are found.*

glossy BRDFs. For example interpolation artifacts were always visible in the icido scene when using the proposed caching scheme. Irradiance caching also depends strongly on the user defined threshold a. If it is set too low, most of the irradiance values are computed and cached which heavily increases the cache size making the algorithm sub-optimal. Even for a low quality (high error) setting as shown in Table 4.1 the number of tests and searches in the octree can be quite high. On the other hand, if a is set too high, interpolation artifacts become visible.

Another problem arises in terms of the algorithm work-flow. The way how to sample the image plane can be crucial because it strongly influences the placement of final gather samples. If we go line by line through each pixel, most samples will eventually only find one valid irradiance cache sample within the error bounds (Figure 4.6).

This yields a visible error since the interpolation becomes a constant step function because each interpolated point in the cached sample's neighborhood gets the normalized weight 1.0 and therefore the same irradiance as the cached sample. The artifacts are perceptible (speckled artifacts) as shown in Figure 4.7 on the left.

As a remedy the image plane can either be sampled stochastically (e.g. using Quasi-Monte Carlo sampling) or progressive refinement. The Quasi-Monte Carlo sampling has not been tested but the progressive refinement scheme. And because the latter still suffers from artifacts too, a different approach using two passes for sampling the image plane has been implemented for reverse photon mapping (see Section 6.7.3). The scheme is also applicable to normal photon mapping and works in the following way:

In the first pass when tracing primary rays from the camera and shooting the final gather rays, we use the simplest frame sampling scheme going line by line through the image plane. At each diffuse primary ray hit point, we query the irradiance cache for re-usable cache entries. If no re-usable cache entries are found, costly final gathering is done and the irradiance cache is

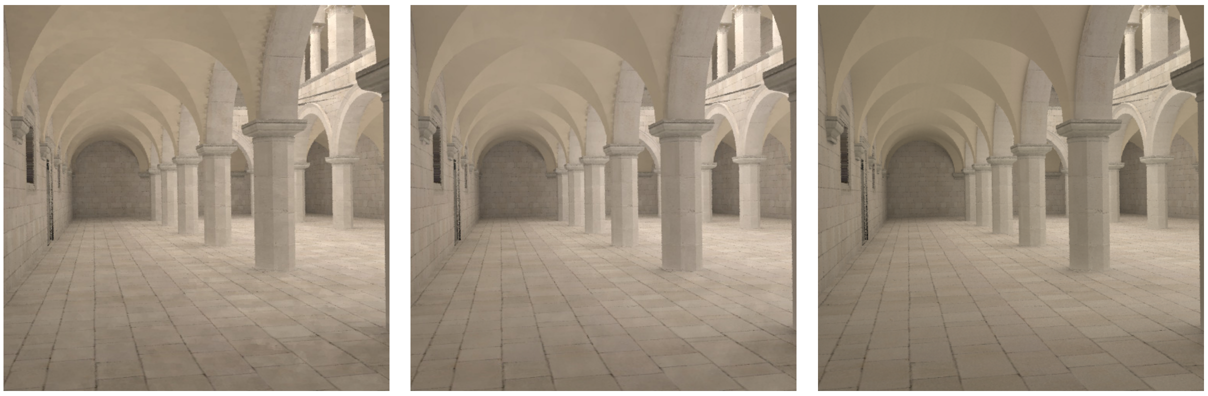


Figure 4.7: *Cached irradiance interpolation using one screen pass (left), two passes (middle), and the reference image (right).*

updated. For all visible hit points, we store a record containing position and normal in 3D and the pixel coordinates. We will refer to this record as pixel sample. The first rendering pass is rather pessimistic since it computes more cache samples than the user-defined error threshold would need. Up to this point, only the incomplete cache that has been build up to the currently sampled pixel is available. There are occasionally superfluous samples that could be interpolated by cache entries which are added to the irradiance cache in consequent lines. This, however, does not harm the algorithm and is even advantageous in the following pass. In the second pass, having the image plane sampled and the irradiance cache filled, we go over all stored non-computed pixel sample records and search for re-usable cache entries in the neighborhood as done in the first pass. This time, we get all re-usable cache entries since the irradiance cache is complete. Moreover, since we have sampled more densely in the first pass than necessary, we find more than one re-usable cache sample in most cases which yields in a cleaner interpolation without the stepping artifacts resulting from the first pass only (see Figure 4.7, middle image).

4.5.2 Radiance Caching

Besides the old algorithm for irradiance caching on Lambertian surfaces [95] that has been used with little modifications till nowadays, several methods have been published in order to make the caching scheme also work with glossy BRDFs. For completeness we will briefly describe the most significant methods.

In realistic scenes not all surfaces are perfectly diffuse. For surfaces with a glossy BRDF simple interpolation of the irradiance across the surface does not work since the light reflection is view dependent and we need to compute the full directional incoming radiance function rather than just a single irradiance value. In order to visualize glossy surfaces one can compute the light transport integral via Monte Carlo sampling of the BRDF. For high frequency, mirror like BRDFs Monte Carlo importance sampling is probably the best choice. However, it is costly for low frequency BRDFs and produces visible noise if not enough samples are used. Hence, one tries to represent the incoming radiance function in a more suitable way with a set of basis functions. Several approaches have been proposed using wavelets [68], spherical harmonics (SH) [64, 73] or hemispherical harmonics (HSH) [20]. Most popular are spherical harmonics (SH) basis functions [52] which are well-suited for low frequency functions defined over a sphere. The incoming radiance as well as the BRDF function are projected onto the basis of (hemi-)

spherical harmonics. Using SH basis functions the light transport integral becomes a simple dot product of the incoming radiance coefficient with the BRDF coefficients. Moreover, the lower the frequencies of light and BRDF the fewer coefficients are needed to reconstruct the original signal. Furthermore, SH allow a GPU-friendly implementation on modern graphics hardware and has recently been used in combination with SH environment maps for real-time rendering [64].

4.5.3 Irradiance Pre-computation

Another possibility to speed up the costly final gathering is to precompute the irradiance at each photon location. This approach was proposed by Christensen [9]. The precomputation can be done in an initial density estimation phase or progressively during the final gather. The former has the advantage of coherence in the search for the neighbor photons at each photon location, since all photons are spatially sorted in a kd-tree structure. Hence, going through each photon location sequentially, results in highly coherent memory access. On the other hand, there might be unnecessary irradiance precomputations in regions that are not visible or areas where the photon density is too high. For these cases, a progressive lazy precomputation might be preferable. The radiance along a final gather ray is then computed by searching for the nearest photon in the neighborhood of the hit point and using (if already precomputed) the photon's precomputed irradiance modulated by the diffuse BRDF. This way, the resulting radiance for a final gather ray corresponds to a piecewise constant approximation of the actual irradiance estimate. However, this is not too problematic because it is adaptive to the photon density and hence to the irradiance. Therefore, the error is low in bright regions which have an important contribution to the final gather and high in (unimportant) low-density regions. Christensen even claimed that the precomputation at every fourth photon location is sufficient. The precomputation of irradiance yields a speedup of one order of magnitude compared to the naive final gathering. The only disadvantage with the precomputation is that it cannot handle glossy BRDFs but only Lambertian surfaces at the final gather ray hit points.

4.5.4 Visual Importance Rendering

Standard photon mapping traces particles from the light sources distributed according to the lights' power distribution and deposits the photons when they interact with surfaces. It performs poorly when little of the lights' total power arrives at query locations important to the final gather. This situation is not uncommon in practice. Indoor environments may have many lights that contribute unevenly to the image. It may also happen that most light path are occluded which is usually the case for large indoor scenes consisting of many separate rooms with a few relatively small light ports like windows and doors. Poor sampling results in excessive noise in the (in)direct illumination estimates derived from the photon map. Moreover, low photon density leads to larger search radii when accessing the photon map, which causes inappropriate samples to be included in the density estimate and therefore severe energy bleeding.
One cause for a poor sampling distribution is the lack of visual importance information because naive sampling from the lights is view-independent and does not consider the camera location. Therefore, several algorithms have been proposed to include visual importance in the photon sampling process. The method of Peter and Pietriek [62] uses an initial pass to emit so-called "importons" (particles traced from the observer). In the following photon tracing pass, they use the previously constructed importon map to compute an importance sampling distribution

from the local importon distribution for each scattering event of a photon. This of course, is expensive due to the cost of computing distributions at every particle bounce and the increased memory storage costs for the importons. Moreover, because this importance sampling strategy is unbiased and requires division by the probability of a certain direction, it results in highly varying photon power especially in areas reached by few photons coming from directions with low probability. Hence, the radiance estimate may be poor.

Another approach to visual importance sampling was introduced by Keller and Wald [43]. They also use an importon map like Peter and Pietriek. However, instead of directing photons into visual important directions building up an importance distribution, they simply decide whether a photon shall be stored or discarded. This decision is computed using Russian roulette with the probability derived from the local importon density at the photon's hit point. Their method produces good-looking results. However, it is very inefficient in generating the photon map. A much larger number of photons than the final number of stored photons must be emitted to get the desired size of the map. Furthermore, the method results also in varying photon power due to the Russian Roulette.

A similar approach was introduced by Suykens and Willems [77]. In their algorithm the photon density is not only driven by the visual importance using importon densities but also by the local photon density itself. Hence, their work introduces the concept of *density control* for photon maps. The algorithm considers the current local sample density in the photon map before storing a new sample and redistributes its power among neighboring samples if the resulting local density would exceed a user-defined density threshold (e.g. $80,000$ photons/m^2). A very similar approach to the latter was used in this thesis and is described in detail in Section 6.9.3.

With the two latter methods, important regions get a dense population of low-power photons while unimportant regions get a sparse population of high-power photons, thus avoiding mixing high and low-power photons. The disadvantage of these two methods is that using importance does not reduce the number of traced photons.

Chapter 5

Data Structures for Density Estimation

In the previous chapters, we have introduced the theoretical subject of density estimation and its practical application for photon mapping. We have seen that photon mapping is generally more demanding with respect to computational resources than statistical density estimation where it is often sufficient to have algorithms of complexity $\mathcal{O}(n^2)$. In photon mapping we cannot afford computing the distance and evaluating the filter kernel for each photon and each density estimation point in 3D since the number of such query points are in the order of 1 to 100 million. Hence, efficient culling of infeasible photons or query points is a key issue in efficient searching for photon mapping and all density estimation techniques. Since the search for feasible photons within the kernel support is often the most expensive part during density estimation, we are now going to investigate data structures for efficient searching.

5.1 Overview

There is a rich literature on spatial data structures outside the computer graphics community [18]. However, in computer graphics already exist many efficient data structures for searching of ray intersections with geometry and polygon clipping. Therefore, first approaches to the problem stored the photons directly on the intersected surfaces of the scene model [71, 88] or accumulated the photons' energy in texels of an illumination texture. These methods are classified as topological dependent methods since the photon storage depends on the geometry and tessellation of the scene model. For these methods we do not need a photon map. In this thesis, we will not deal with such approaches but focus on topological independent density estimation methods where photons are stored loosely in 3D space in a so-called *photon map*.

Any data structure suitable for searching can be used for the photon map. The simplest among those is the unsorted list or array where every element needs to be tested. Despite its simplicity the list performs best for a small number of elements (up to ≈ 100), which can be exploited for more sophisticated data structures such as the grid or kd-tree. The *grid* subdivides the domain into regularly spaced cells called voxels, which can be localized in constant time. Therefore, a grid is a fast but not adaptive search structure. It is further described in Section 5.3. A data structure that is better at handling non-uniform distributions of photons is the *binary space partitioning tree* (BSP-tree) in particular the axis aligned BSP tree referred to as *kd-tree* [67]. However, it might also be useful to investigate an *octree* structure (8 children per node). Since the kd-tree is the most popular data structure, it is used throughout the thesis and

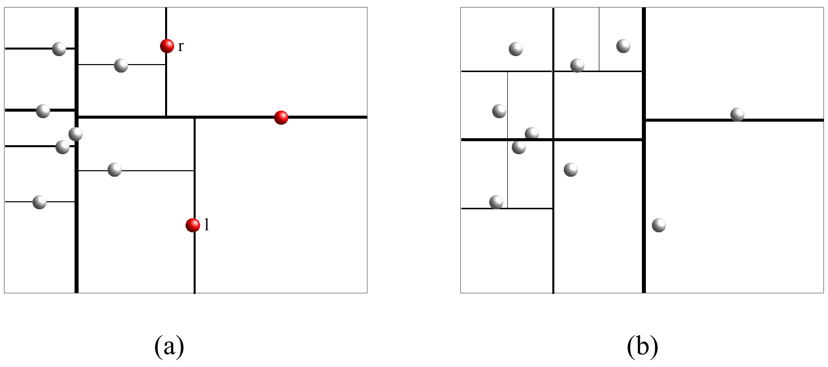


Figure 5.1: *Photon median balanced kd-tree (a) and the spatial median balanced kd-tree (b). Note that the depth of the spatial kd-tree is greater than for the balanced one.*

described thoroughly in the next section. Another interesting and efficient but more complex data structure is the *Voronoi diagram*, which we will only mention shortly.

In a Voronoi diagram each node (photon) is linked to its direct neighbors. For a nearest neighbor query, we first need to identify the node occupying the cell of the query point. This can be done via a directed walk towards the query point starting from a random node (e.g. chosen from the voxel of a grid on top of the diagram). The nearest neighbors are then trivially found via a seed-growing-like algorithm. The Voronoi diagram supports queries for the k-nearest neighbors in $\mathcal{O}(k \log n)$ time but requires $\mathcal{O}(n^2)$ storage space in three dimensions. Nevertheless, the Voronoi diagram is interesting in the way it can provide us information about the density and local distribution of the points (e.g. if the occupied area of an associated node is the same for all nodes, the density is uniform).

5.2 Kd-tree

The kd-tree is probably the most commonly used data structure for searching since it is fast and memory-friendly due to the adaptation to the data density. There are different layouts and various ordering schemes for kd-trees. We will compare several ordering schemes and layouts for kd-trees over photons.

The traditional layout of the kd-tree proposed by Jensen [33] stores the photons with the nodes of the tree where each splitting plane is defined by the location of the median photon of the current node (see Figure 5.1.a). The photons are stored in an array that is organized as a heap (i.e. the left child is stored at index $2 \cdot i$ and the right child at $2 \cdot i + 1$, if the current index is i). This ordering scheme requires that the kd-tree is left balanced such that each interior node has at least a left child but not necessarily a right one. Furthermore, the tree is constructed up to the leaves and hence each interior node has between one and two children. Since the tree is balanced, the maximum depth is $\log n$ where n is the total number of photons.

Within the framework of *reverse photon mapping* which is part of this thesis and is described in Chapter 6, we have also analyzed other kd-tree layouts for searching. Since the number of

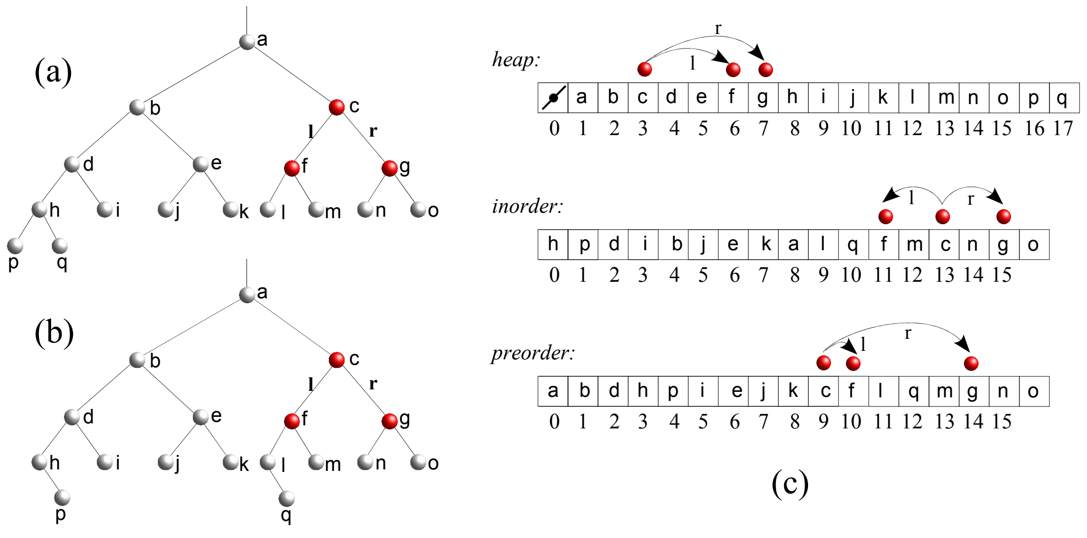

(a)
(b)
(c)
heap:
inorder:
preorder:
a b c d e f g h i j k l m n o p q
h p d i b j e k a l q f m c n g o
a b d h p i e j k c f l q m g n o
l
r

Figure 5.2: *The figures on the left show a photon map representation using a left-balanced kd-tree (a) and a balanced kd-tree (b). The figures on the right (c) show possible memory layouts for the photon-nodes representing the tree in (a) or (b). The heap order (top) was proposed by Henrik Wann Jensen. The tree must be left-balanced (a) for the heap representation. For the inorder and preorder scheme the tree has to be balanced (b). The preorder scheme needs two additional bits in each photon-node to indicate if the node is a leaf-node, an interior-node, or a one-child node.*

”particles” in the tree is much larger for *reverse photon mapping* than for ”normal” photon mapping, the standard layout *à* la Jensen is not appropriate for our purposes. First of all, the time for constructing the tree is unacceptable and even the search does not perform better than our proposed method. Therefore, we experimented with alternative layouts for the kd-tree.

The first tests are related to Jensen’s kd-tree proposal in the way that each photon corresponds to a node in the tree and lies on the axis-aligned splitting plane of the node. The difference is in the ordering scheme of the tree in the memory. Jensen proposed a heap-like order that corresponds to a breadth-first layout: the tree is stored from the root to the leaves such that each depth level of tree comes after the other in the array. Hence, tree traversals are most memory-coherent at the top level of the tree but far jumps are necessary to the lower levels of the tree (e.g. to the leaves).

A simpler ordering is the *inorder* scheme where the array of photons is organized in the same way the tree is constructed. This means the root is the middle of the array in range $[i_l, i_r]$ at index $i_c = (i_r + i_l)/2$ and the children are located at $(i_l + i_c - 1)/2$ and $(i_r + i_c + 1)/2$ where i_l is the index of the leftmost photon in the sub-tree of root node i_c and i_r is the rightmost (see Figure 5.2). This scheme can be thought of as a projection of the kd-tree to the 1D array of photons (i.e. the tree collapses to a line). Consequently, the layout has best coherence at the lower levels of the tree since neighboring photons in space are also neighboring elements in the array. However, in the upper levels, at the root of the tree, traversals are most incoherent as the indices of the child-nodes are very different.

Another possible ordering for the array of photons is the *preorder* scheme (see Figure 5.2). The tree is stored in the order: root, left child, right child. Thus, the left child (if any) follows its root node at index $i_r + 1$ and the right child is always located at index $i_r + (n_r - n_r/2)$, where n_r is the number of nodes (i.e. photons) in the sub-tree excluding the root node with index

i_r. This shows that the tree is right-balanced and a right child always exists unless the current node is a leaf node. Despite all assumptions, this preorder scheme is more efficient for searching than the inorder scheme. It is also slightly faster than Jensen's left-balanced heap-layout, which however might be due to implementation differences.

Other ordering schemes have not been tested in the scope of the thesis. However, using these three schemes we stress that we have covered the most important ones. And anyway the ordering scheme does not have that much impact on the search performance as the tree structure and the spatial layout of the tree have, which we will focus on next.

5.2.1 Our proposed Kd-tree8

Using a balanced heap-like tree representation as suggested by Jensen, we avoid the storage of two pointers per node. We will see that we actually need only one. Since each node is represented by a photon, which uses at minimum 20 Bytes, the tree becomes very large for several million photons and might eventually not fit in the main memory. Hence, the memory-footprint of a tree traversal can become considerably large since unneeded data from the photon-nodes (power, 3D position, incoming direction) is read to the CPU cache. We have therefore developed a new tree representation that does not explicitly store photons in the nodes but uses specific traversal nodes which store merely the minimum information needed for a tree traversal. Due to the smaller size of our kd-tree (3-4 times less memory than the kd-tree layout used by Jensen [34]), we reduce the memory foot-print of a traversal which helps to increase cache coherency. The actual photons are kept in the sorted array and are only indexed by the leaf nodes of the kd-tree. This way we need more memory in total because each traversal node needs to store at least the axis aligned splitting plane (which Jensen avoided by exploiting the position of photon nodes) but we gain more flexibility with respect to the shape of the tree and its spatial layout. So, it is possible to use either a photon-median (balanced) or a spatial-median layout. However, the spatial-median is preferable and the photon-median has only been used for comparison. The difference in the performance can be observed in Table 5.2. The spatial-median layout has several advantages:

1. Construction of the tree is faster since we simply fix the splitting plane of a tree-node to the middle of the longest side of its bounding box and avoid a k-median sort of the photons.

2. The search performance is improved even if the depth of the tree is increased, especially for none-uniform, skewed distributions (e.g. caustics).

3. Construction can be combined with the sliding-midpoint rule which further improves the search performance.

4. The tree adapts to the local density of photons. Hence, using a node's voxel size and its associated photons, we can draw rough assumptions about the local density that can be exploited for fast bandwidth selection in pilot estimates or as an upper bound for the k-nearest neighbors search radius. Moreover, fluctuations in the density of photons (which is decisive in bandwidth selection) are reflected in the depth variations of the tree.

The only disadvantage of the spatial-median layout is that we need slightly more memory since the tree is not balanced anymore and an index or offset to one child must be stored per node. In addition to the spatial median sorting, we apply the sliding-midpoint rule:

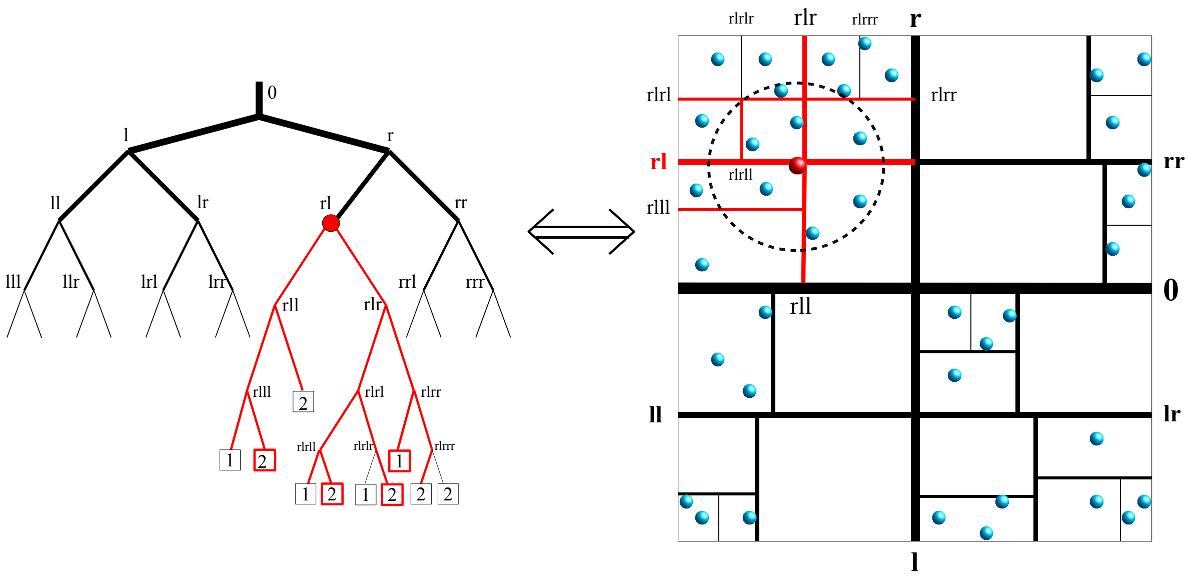


Figure 5.3: *The spatial layout in 2D (right) of the kd-tree with sliding mid-point rule (left). The sample query shows the tested interior nodes (red lines) and leaves (red boxes). Here, the maximum number of photons per leaf is 2. Note, how the sliding-midpoint rule shifts the plane to the nearest photon location to cut away the largest possible empty space!*

- set the splitting plane to the middle of a node's bounding box aligned with the axis of longest side,
- sort all photons associated with the node to the left or right side of the slitting plane,
- if either the right or left halfspace does not contain any photons, slide the splitting plane to the nearest photon location in the non-empty halfspace.

This sliding-midpoint rule is a key issue for good performance of the kd-tree. Although such rule looks quite simple, it was proven to be bounded in worst case by a poly-logarithmic time for approximate search queries [14, 53]. An example for a visualization of our kd-tree in 2D with spatial layout and sliding-midpoint rule is shown in Figure 5.3. We call our tree structure *kdtree8* because each node of the tree has 8 Bytes, which are defined as shown in Figure 5.2.1. The variable `plane` stores the splitting plane position aligned with the axis that is stored in the lower 2 Bits of `rightOffset`. Given the bounding box of the tree, the variable `plane` could be avoided if we restrict the tree layout to a fixed spatial-median. However, we want to keep the tree most flexible to allow for sliding-midpoint rule based construction as well as for photon-median balanced construction. Therefore, we need to store the plane position and the index or offset to at least one of the two child-nodes. Since we have chosen to use the *preorder* memory layout, we only need to store the offset to the right child-node because the left child-node is always found at index $i+1$. Using the spatial layout, the tree is not necessarily regular anymore and there happen to be empty leaves in the tree that do not contain any photons. Furthermore, we apply the sliding mid-point rule to cull away empty parts of the tree as soon as possible. Thus, we need to insert special nodes into the tree that simply indicate that the sub-tree ends there. In the same way, we need to identify whether a node is a leaf or an interior-node of the tree. In case of an interior node, we continue the tree traversal. In case of a leaf-node the plane

```
struct InteriorNode
{
    float plane;              // axis-aligned splitting plane
    unsigned int rightOffset; // offset to the right child
    enum EType {
        EE_LEAF  = 0x0,
        EE_LOAD  = 0x4,
        EE_EMPTY = 0x8,
        EE_INNER = 0xC };
    enum EMask {
        EE_TYPE  = 0xC,
        EE_PLANE = 0x3 };
};
```

is used to store the precomputed bandwidth (squared search radius) and the upper 28 bits of `rightOffset` are taken as the index into the array of photons.

The important issue of the separate 8-Byte-node tree is that a single leaf can point to a contiguous sequence of photons in the array. The maximum number of photons per leaf is limited to a constant number T_m (e.g. $T_m = 16$). Since all bits of the leaf-node are used, the number of elements per leaf is stored in the photon structure itself since this structure contains one unused Byte due to alignment. This allows us to store the number of elements in the first photon pointed to by the leaf-node and enables up to 256 photons per leaf which is sufficient. The type `LOAD` is a special type of nodes that is only used by the external tree which is descibed in *reverse photon mapping* in Section 6.10. The pseudocode for the recursive top-down kd-tree construction starting at the root node for the whole photon array in range $i_{from} = 0$ to $i_{to} = n$ is shown in Figure 5.4.

Since we store several photons per leaf, the *total* memory needed for the *kdtree8* (including the photon array) is only slightly higher than for the tree layout with photon-nodes. By using the kd-tree only for traversal to the leaves, we limit the test for nearest photons to a minimal spatial region around the query point rather than along the entire kd-tree traversal.

Storing several photons per leaf results not only in an improvement of the search performance, but also in a substantial decrease of construction time and memory space since fewer tree nodes are created. The effect of having several elements (photons) in the leaves of a kd-tree was studied by Talbert and Fisher [79]. They conclude that 10 to 30 photons does not decrease performance but reduces the memory requirements of the tree by up to a factor of $\log_2 30$. Since we stop the recursive sorting if the number of photons in the sub-tree falls below the threshold T_m, the time complexity is reduced by $\mathcal{O}(N \log_2(T_m))$. Hence, the time complexity of the kd-tree construction is $\mathcal{O}(N \log_2 \lceil N/T_m \rceil)$ where N is the total number of photons. In our test case, having maximal 32 photons per leaf, the memory required by the nodes of the kd-tree is about 30 to 60 times less than the memory for the photons. Hence the whole tree can be stored in the main memory of a standard PC even for dozen millions of photons. Moreover, we experienced that even more than 30 photons per leaf does not decrease performance of searching. The impact of the number of photons per leaf on the search timings and construction timings is shown in Table 5.1.

```
function constructTree($i_{from}, i_{to}$) $\longrightarrow N_{nodes}$
  $N_{ph} \leftarrow i_{to} - i_{from} + 1$
  if $N_{ph} = 0$ then
      create empty node
      return 1
  end if
  if $N_{ph} \leq N_{leaf}$ OR tree depth $\geq D_{max}$ then
      create leaf node with index $i_{from}$ and offset $N_{ph}$
      return 1
  else
      create interior node $H$ and keep index $i_H$
      select axis $a$ with longest side of the current voxel
      compute [spatial $\|$ photon] median position $P_H$ along $a$
      binary sort the array $A_{ph}$ in $\mathcal{O}(n)$ time so that $\forall(i,j)$ with $i,j \in [i_{from}, i_{to}]$ and $ph[i].p < P_H \wedge$
      $ph[j].p \geq P_H \Rightarrow i < j$
      {if either the left or right half space given by the splitting plane $P_H$ does not contain any photons,
      move $P_H$ to the nearest photon position on the right respectively left (sliding-midpoint rule)}
      $N_L \longleftarrow$ constructTree($i_{from}, i_{left}$) {recurse for the left half space and returns number of nodes
      $N_L$}
      $N_R \longleftarrow$ constructTree($i_{right}, i_{to}$) {recurse for the right half space and return number of nodes
      $N_R$}
      initialize $H$ with rightOffset $\leftarrow N_L + 1$, plane $\leftarrow P_H$, and axis $\leftarrow a$
      return $N_L + N_R + 1$
  end if
```

Figure 5.4: *The construction of our spatial kd-tree with sliding-midpoint rule.*

Atrium scene (glossy/diffuse)						Sponza scene (diffuse)					
N_{leaf}	N_{nodes}	$\bar{V}_{nodes}$	$\bar{V}_{leaves}$	T_c	T_s	N_{leaf}	N_{nodes}	$\bar{V}_{nodes}$	$\bar{V}_{leaves}$	T_c	T_s
1	38,666,700	13,195	4,941	14.73	10.75	1	17,486,267	7,587	2,833	6.11	3.00
2	22,648,498	7,617	3,263	12.87	8.50	2	10,064,441	4,422	1,882	5.59	2.45
4	12,106,679	4,031	1,767	11.50	6.95	4	5,266,633	2,338	1,059	4.94	1.67
8	6,278,929	2,098	953	10.62	6.08	8	2,659,460	1,207	548	4.27	1.63
16	3,265,263	1,100	501	9.87	5.56	16	1,345,430	629	283	4.23	1.52
32	1,739,938	591	253	9.35	5.33	32	682,949	333	144	4.02	1.39
64	952,313	329	135	8.88	5.23	64	344,390	182	78	3.81	1.39
128	443,773	187	71	9.39	5.38	128	173,284	104	40	3.65	1.44
256	217,547	113	42	8.03	5.34	256	88,694	63	24	3.25	1.44

Table 5.1: *Construction time (T_c) and fixed-radius search time (T_s) in seconds for a different number of photons per leaf (N_{leaf}) in the proposed spatial kd-tree with 8-Byte-nodes and preorder depth-first layout constructed using the sliding-midpoint rule. The table shows also the average number of accessed nodes per search query ($\bar{V}_{nodes}$) in the kd-tree (including leaves) and the average number of accessed leaves per query ($\bar{V}_{leaves}$). We have tested two scenes: the atrium scene which is rather glossy and the sponza scene which is almost diffuse. The scenes are shown in Section 6.21. The tests in the left half are carried out for the atrium scene with approx. 16.5 million photons ($\approx 3,100$ photons per query). The tests in the right half are carried out for the sponza scene with approx. 7 million of photons ($\approx 1,600$ photons per query). Both search times are the result from 10,000 successive search queries including density estimation. Note that the maximum number of photons per leaf is limited by one Byte allowing for 256 photons.*

5.3 Grid

The largest part of the rendering time is spent in searching the nearest neighbors. Therefore, any implementation that is faster than the standard kd-tree would yield in a significant speedup. The regular grid is one such data structure. It has been successfully used in several photon mapping papers [86], and theses [26, 32]. A photon grid is a regular sub-division of the photon bounding box into equal-sized cells called voxels. The grid needs additional memory for representing the voxels, which is usually only one index or pointer per voxel to the photon array. Nevertheless, the memory overhead for millions of voxels can take a considerable amount of memory in addition to the photon array. Since the grid is not adaptive, it may perform badly if the distribution of photons is far from uniform (as for caustics) because this would result in few voxels with a high number of photons and many voxels which are empty (usually more than 75% for most scenes [26]) and more than 3/4 of the memory is wasted. Nevertheless, the grid performs usually better than a kd-tree due to its constant search complexity for the nearest voxels. However, the performance depends heavily on the resolution of the grid. Too large voxels decrease performance since many one to one comparisons inside the voxels have to be tested. Too small a voxel size rapidly increases the number of empty voxels (with $\mathcal{O}(n^3)$). The construction of the grid is the same as for the hybrid grid described next except that we do not further sort the photons inside a voxel. The search in a grid is similar to a 3-dimensional seed filling. In case of the k-nearest neighbor search, we start to compute the distance to all photons inside the current voxel containing the query location and proceed with the neighboring voxels (which can be found in $\mathcal{O}(1)$ time) until we have found the k-nearest photons or reached the farthest voxels in the maximum search radius. Detailed code for the regular grid is omitted, since it is not relevant for this thesis and has not been investigated further. For implementation details, refer to [86, 26].

5.3.1 Hybrid Grid

A grid is fast but not adaptive. The kd-tree is adaptive but less efficient. So, we can combine them to benefit from both data structures. Two combinations are possible: a grid in each leaf of the kd-tree, or a kd-tree in each voxel of the grid. However, only the latter combination has been implemented, which works as follows. We utilize a hybrid grid with a regular (coarse) resolution and a secondary (adaptive) data structure in each voxel. This is preferable over the standard grid where every voxel contains only a simple list of photons. A hybrid approach depends less on the resolution and wastes less memory due to fewer voxels. Moreover, the search complexity for a hybrid grid with many small localized kd-trees in its voxels is better than for a single large tree since we only need to account for photons in a small fraction of the scene, whose extend can be determined in $\mathcal{O}(1)$ time. Therefore, combined with our spatial kd-tree8, it naturally performs better than using a single instance of the tree. In some sense, this is related to the idea of having multiple photon maps as proposed by Larsen and Christensen [46]. However, combined with the aggregate search described in *reverse photon mapping* in Section 6.9.2, we gain 30% to 65% speedup for the kd-tree8 compared to the traversal of individual queries. The traversal of several coherent queries at once is more efficient for a single large tree. Therefore, we claim that the tree is still preferable over the grid for coherent search queries. Nevertheless, the hybrid grid is still an interesting alternative to a single kd-tree and should be investigated further.

The construction of the hybrid grid is very similar to the normal grid construction except for the treatment of a leaf. For a standard grid, the leaves contain a list of photons which have to be checked one by one. For the hybrid grid, a leaf contains another photon map of any type. Therefore, we need fewer voxels than for the normal grid and the computation of the resolution can be simplified to the following proposed algorithm:

- compute the resolution of the grid as usual but set the minimum size of a voxel to the maximum gather radius for density estimation (this ensures that only the direct neighboring voxels (26 neighbors) must be tested and eventually queried during density estimation
- allocate an array of pointers to an abstract photon map type which can be a list or a certain kd-tree (or a grid itself)
- recursively subdivide the domain (until a single voxel is left) in the same way as for the spatial kd-tree8 but with discrete splitting plane locations aligned with the voxel boundaries
- if the recursion reaches a single voxel, and the number of associated photons is larger than a minimum threshold (70), create a new photon map of user-defined type (e.g. list, kdtree8, balanced kdtree).
- construct the new photon map for the photons in the current voxel's bounding box.

Instead of bounding the minimum size of a voxel to the maximum gather radius R_{max}, we could also set it to $2 \cdot R_{max}$, which would limit the search to only 8 voxels as proposed by Wald et al. [86]. However, we experienced that the resolution of the grid in the case of $2 \cdot R_{max}$ can be too coarse (8 times less voxels!) if R_{max} is relatively large.

The search in a hybrid grid at a query point p_q with search radius R can then be formulated as follows:

- find the voxel v, where p_q is located in, in $\mathcal{O}(1)$ time and query its photon map
- compute and insert the indices of all direct neighbors of v to a list of max. 26 entries (not at the boundary of the grid!)
- compute the distance d_i from p_q to all 6 sides $i \in [1..6]$ of v
- if d_i is greater than R remove all 9 voxels on side i from the list
- do the same for all max. 12 edges and remove all 3 voxels from the list connected to the edge j (if any) if d_j^2 is greater than R^2.
- finally, do the same procedure for the max. 8 corners of the remaining diagonal voxels (if any)
- successively query the photon maps of the remaining voxels in the list (between 0 and 26)

Notice that in only few cases all 26 neighboring voxels must be queried. In most cases only 1 to 8 neighbors need to be tested since the search radius is on average much smaller than the maximum search radius.

Scene	Data Struct.	Type	Layout	N_{leaf}	$T_{construct}$	T_{search}
Atrium	kdtree	jensen	balanced preorder	1	14.90	10.20
Atrium	kdtree	jensen	balanced inorder	1	14.50	12.70
Atrium	kdtree8	our	balanced preorder	1	16.80	9.48
Atrium	kdtree8	our	balanced preorder	16	12.66	5.67
Atrium	kdtree8	our	spatial preorder	16	9.87	5.56
Atrium	kdtree8	our	spatial preorder	32	9.35	5.33
Atrium	grid	normal	spatial + list	-	5.66	8.56
Atrium	grid	hybrid	spatial + balanced kdtree	1	12.22	10.10
Atrium	grid	hybrid	spatial + kdtree8	16	10.20	5.47
Sponza	kdtree	jensen	balanced preorder	1	5.82	2.96
Sponza	kdtree	jensen	balanced inorder	1	6.07	3.66
Sponza	kdtree8	our	balanced preorder	1	6.72	2.59
Sponza	kdtree8	our	balanced preorder	16	5.32	1.61
Sponza	kdtree8	our	spatial preorder	16	4.23	1.52
Sponza	kdtree8	our	spatial preorder	32	4.02	1.39
Sponza	grid	normal	spatial + list	-	2.33	1.91
Sponza	grid	hybrid	spatial + balanced kdtree	1	5.25	2.69
Sponza	grid	hybrid	spatial + kdtree8	16	4.60	1.41

Table 5.2: *The construction times $T_{construct}$ and query times T_{search} for different search data structures and layouts. A fixed radius was used for the search. From left to right column: The scene used for testing, the search data structure, its type and layout, the maximum number of photons per leaf (voxel) N_{leaf}, the construction time, and the render time (search + density estimation time) for 10,000 successive queries. The total number of photons is about 16 million for the atrium scene and 7 million for the sponza scene. The type of the kdtree indicates whether it is the original layout of the kdtree proposed by Jensen where each photon corresponds to a node of the tree or whether it is our proposed kdtree with 8-Byte nodes where the photons are stored in the leaves only. The hybrid grid consists of 11,700 voxels (atrium) and 46,128 voxels (sponza) each containing a kdtree (or list) of same type and layout.*

5.4 Results

A full comparison of the performance of our *kdtree8* relative to the other mentioned search data structures is shown in Table 5.2. It shows the timings for construction and rendering for the different kd-trees with various layout schemes as well as a grid and hybrid grid structure. The first two rows correspond to the balanced kd-tree with one photon per leaf as proposed by Henrik Wann Jensen. The next rows show the timings for our proposed spatial kd-tree with 8-Byte nodes and sliding mid-point rule where the photons are only stored in the leaves. We have tested both the photon-median balanced version and the spatial-median balanced variant. For comparison purposes, we show only the difference between 1, 16, and 32 photons per leaf. An exhaustive analysis for various leaf settings is shown in Table 5.1. The time T_{search} includes the time for density estimation and statistics which is about 30% to 40% of the total time T_{search}. All trees are implemented in a recursive way for construction as well as for searching. An iterative version could decrease the search time further by up to 10% [9].

Chapter 6

Reverse Photon Mapping

Photon mapping (PM) is the commonly used global illumination algorithm since it yields good-looking images without the distracting noise of unbiased Monte Carlo algorithms. Moreover, Monte Carlo algorithms such as path tracing converge very slowly to a visual acceptable solution. The reason for this is that PM efficiently re-uses light information and performs a convolution of illumination across all surfaces filtering the high frequencies. This works well since indirect illumination changes slowly in most cases. Hence, PM speeds up rendering of global illumination by having more information at hand on the expense of using more memory.
In our approach, we build on this principle but exploit much more the modern PC's memory. Instead of storing only the incoming illumination on surfaces, we also store information about indirect visibility. In normal PM one usually prepends a Monte Carlo sampling step referred to as final gathering before estimating illumination from the photon map. However, each final gather ray is only used once for computing the radiance from the photon map and thrown away afterward. Thus, FGRs that intersect the same surface will eventually use a similar set of photons for computing the radiance from the photons energy. We propose an algorithm that keeps all final gather rays (FGRs) and organizes them in an efficient data structure for searching similar to the photon map. We have presented and analyzed several data structures suitable for density estimation in Chapter 5. Keeping all FGRs, we don't need to compute the radiance for each FGR separately but can turn the procedure around and distribute a photon's energy to many FGRs at once. This concept is algorithmically advantageous due to the fact that the number of final gather rays is usually much larger than the number of photons and we will save queries compared to normal photon mapping. The achieved speedup is in the order from 150 to 400 percent. On the other hand, the memory storage needed for storing all final gather rays and photons becomes huge (beyond the capacities of standard PCs). As a remedy, we will show different techniques such as image tiling and external caching to a hard disk. The rendering equation for our reverse photon mapping is very similar to normal photon mapping but uses a slightly different density estimation technique which we will describe in Section 6.4.

6.1 Previous Work

The time consumed by diffuse indirect illumination is typically the most significant part in the whole rendering. This issue has been addressed in several papers. Here, we focus on a method that aims at high quality solutions not allowing approximation artifacts that is comparable to normal photon mapping with final gathering. We will exclude methods based on radiosity as well as other techniques that explicitly work with polygonal representations. The recently developed real-time global illumination techniques such as Instant Radiosity work only for diffuse surfaces and are not general enough to be used for high-quality rendering taking arbitrary light transport into account.

The probably most prominent and oldest acceleration technique for indirect illumination computation is the irradiance cache [95] and [94]. This solution works efficiently for slowly changing indirect illumination on diffuse surfaces. Irradiance caching is excessively explained in Section 4.5.1. The irradiance cache normally accelerates by one order of magnitude depending on the user defined error threshold. However, it can introduce visible artifacts and the user parameters need to be tuned in order to obtain nice results in acceptable times. This can be cumbersome and needs experience.

Christensen proposes to precompute the irradiance at the positions of one quarter of all photons. While the storage space required for the photon map is increased, the irradiance for each FGR is approximated by a piece-wise constant function that forms a 2D Voronoi diagram. Instead of computing the density from the k nearest photons, the algorithm searches for the closest photon with a "similar" normal to the one at the final gather ray hit point and returns the irradiance that was precomputed at its location. This technique offers a speedup of around one order of magnitude depending on the number of queries to the photon map. However, the speedup varies strongly. The quality of the results depends on the scene and user settings. To avoid artifacts the user is required to adjust parameters such as "similar enough" for photon normals.

Christensen and Batali [10] proposed the concept of an irradiance atlas as an approximative representation for diffuse indirect illumination. They associate photon maps with objects in the scene and construct a hierarchical "irradiance brick map" represented by an octree over the entire scene. The depth of the brick map adapts to the density and normal deviations in the photon maps. A brick consists of 8^3 voxels for which a constant irradiance value is precomputed. The brick map is capable of storing a large number of photons since the brick data is stored to disk and only cached to main memory during access. Despite the large number of photons, the method is not able to eliminate the noise when rendering the directly visible irradiance from the brick map. The results look still noisy and blurry. And hence, a final gather step is needed which boosts the render timings to approximately 4 hours for a high quality image.

A photon mapping method based on kernel density estimation (KDE) is described in the paper from Lavignotte and Paulin [50]. They propose a two pass algorithm for fast photon mapping using graphics hardware. Each photon that hits a diffuse surface splats its contribution to the pixels within its projected footprint (i.e. projected disc) in the frame buffer using a box kernel. They account for adaptive bandwidth selection in each pixel and deal with boundary bias for triangular meshes. However, their approach is limited by the precision of the graphics hardware and does not work with arbitrary topology. Since the KDE is restricted to a 2D projection of the photon's footprint in the image plane, occlusion and topological bias is enhanced for non-planar surfaces. Moreover, the density estimation is limited to a box kernel function and further bias is introduced due to the quadrilateral kernel support (projected quad) rather than ellipsoidal (projected disc).

In this thesis we present our acceleration technique for photon mapping that has been published to Eurographics in 2005 [29]. The method is also based on kernel density estimation but utilizes final gathering to obtain high quality results. It yields satisfactory and robust results without the need for playing with parameter settings. We achieve this at the expense of using more memory. Our algorithm can also be combined with several acceleration techniques such as irradiance caching, density control, and importance sampling.

6.2 Overview

The basic idea of our algorithm is to reorganize the computation of density estimation for diffuse indirect illumination in the reverse order. We will refer to normal photon mapping (NPM) as the way how the global photon map is used traditionally [34]. In NPM, photons are traced first. The photon map is constructed and final gather rays (FGR) are shot for every primary ray hitting objects in the scene. At each FGR hit point, the irradiance is computed by density estimation based on the k-nearest neighbors (kNN) search. The detailed photon mapping algorithm is described in Chapter 4 on page 35.

We propose a *reverse photon mapping* algorithm that uses the same set of FGRs and the same set of photons as for NPM. The difference is in the order of the density computation. For each photon we find all the hits from FGRs that we refer to as reverse photons. We distribute the photon's energy to all contributing reverse photons and their corresponding pixels. The reverse order results in faster computation. The analogy of reverse photon density estimation to NPM is shown in Figure 6.2. Below, we describe the simplified version of the proposed algorithm without issues related to caching.

6.3 Data structures

In the first step we sample the image plane and shoot FGRs at all primary ray hit points. We will refer to the visible hit points of primary rays as *pixel samples*. For each pixel sample, we store a record containing several attributes that we will describe later. At each pixel sample we shoot final gather rays across the hemisphere and store reverse photons corresponding to their hit points with scene objects. The reverse photons are similar to the concept of importons [62]. However, since we use them in a different context, we prefer another name to distinguish between these concepts. The reverse photons are stored in an array A_r. The reverse photon contains the information listed below in Figure 6.1.

```
struct CReversePhoton
{
    CVector3D position;        // position of FGR hit point (12 bytes)
    unsigned char theta, phi;  // incoming direction - compressed to 2 bytes
    unsigned short flags;      // 1 byte for flags + 1 byte alignment
    CVector3D contribution;    // reverse photon weight for 3 spectra (12 bytes)
    unsigned int sampleIndex;  // index to the pixel sample array (4 bytes)
};
```

Figure 6.1: *The reverse photon record (not optimized!)*

The storage of an reverse photon requires 31 Bytes, however, for better alignment in the memory 32 Bytes are used. The member variable *flags* has different meanings depending on the search data structure we use. For the original balanced kd-tree *à* la Jensen [34], it is used for storing the dimension of the splitting plane (x, y, z). Our proposed spatial kd-tree uses it to save the offset from the first to the last element in a leaf. In addition to photons, the reverse photon contains an index to its originating pixel sample. Moreover, instead of storing energy it stores the weight c of the eye path from the corresponding pixel to the final gather ray hit point for three wavelengths (here just RGB or XYZ). Since the weight c results from BRDF sampling, it is reciprocal. This means we are allowed to revert the eye path and consider it as the light path describing the radiance transfer from FGR hit point (reverse photon) to the eye.

After having sampled the image plane and stored all reverse photons, we construct the reverse photon map which accelerates searches of the kind: nearest neighbor, k-nearest neighbors, and all neighbors in radius R (used for KDE).

6.3.1 Kd-tree Construction and Layout

In the second step, we construct an efficient data structure for fixed-radius search over reverse photons. We have tested several data structures such as the original balanced kd-tree [34] in various memory layouts (inorder, preorder, heap order), a hybrid grid where the voxels contain any other data structure, and a simple list for comparison, see Chapter 5 for details. We evaluated as the most efficient a spatial kd-tree with preorder layout and sliding mid-point rule. Unlike the original balanced kd-tree where the interior nodes and leaves correspond to single photons each storing an axis-aligned splitting plane (4+1 Bytes), the reverse photons are only stored in the leaves and are pointed to by the kd-tree nodes. This way, both, the interior nodes and the leaf nodes of our kd-tree take only 8 Bytes. Similar approach was used by Wald et al. [85]. The total memory consumption of our search data structure including reverse photons is only slightly higher since the kd-tree is around $T_m * (32Bytes/8Bytes)$ times smaller than the array of reverse photons, where T_m is the average number of reverse photons per leaf. For example, if $T_m = 32$ the kd-tree is up to 128 times smaller than the array of reverse photons. Therefore, the whole tree fits into the main memory of a standard PC even for a high number of reverse photons. However, the highest performance gain yields the kd-tree construction since the time for the tree construction is proportional to the depth of the tree and the depth is reduced by up to $\log_2(T_m)$.

6.3.2 The Dual-Tree for Coherent Accesses

In addition to the reverse photon map, we also construct a second kd-tree, which we will call the *kd-tree over photons*. The procedure is almost the same as for reverse photons. We shoot the photons from the light sources and store them in the photon array A_p when they hit a diffuse surface. This is exactly carried out as for normal photon mapping with the exception that we do not use the photon map for searching. However, when building up the *kd-tree over photons*, we benefit from the spatially sorted photons. Queries formed by sequentially traversing the photon array A_p, establish spatial and therefore memory coherent accesses to the reverse photon map A_r. The queries are highly coherent because both trees are sorted spatially. This allows us to use several optimizations such as the *aggregate search* (see Section 6.9.2) and the external cache (Section 6.10). The *kd-tree over photons* uses the same data structure as the reverse photon map where the photons are stored in the leaves and the kd-tree on top consists of 8 byte nodes. This has the advantage that we can simply throw away the kd-tree after construction and applied density control (Section 6.9.3) which saves us memory. Furthermore, the spatial layout of the kd-tree over photons can be exploited in various ways as we will show for the case of adaptive kernel density estimation (Section 6.5).

6.4 Kernel Density Estimation

We have built up the kd-tree over reverse photons and the kd-tree over photons. Therefore, we have highly coherent queries in the reverse photon map. Using *kernel density estimation* (KDE) we search at every photon position for all reverse photons not farther than the radius R_j. The computation of R_j is described in Section 6.5. Recall that in normal photon mapping the k-nearest neighbors (kNN) density estimation is usually used. The kNN density estimation is an adaptive method and has been proved to be robust while still efficient and easy to implement. The number for k is usually provided by the user. In combination with final gathering, k is set to smaller values ($\approx 30 - 100$) to avoid occlusion bias (light leakage) and blurring in low density regions. As a rule of thumb, we set it to $\sqrt[3]{N}$ where N is the total number of photons (see Section 4.4.2). However, kNN density estimation is far from optimal [74]. It results in discontinuities in the estimate and suffers from heavy tails in low density areas. See Chapter 3 for details. In our algorithm we use *kernel density estimation* (KDE) with adaptation to the local density which is often referred to as *variable kernel density estimation.* Variable KDE produces slightly better results than kNN density estimation since it better preserves high density gradients which are strongly blurred by kNN density estimation. The KDE methods are well studied in statistics [74] [89].

Different kernel functions are supported by our density estimation method. Using an arbitrary kernel (Box, Cone, Epanechnikov, Gaussian, and Biweight) we distribute the flux after multiplying by the weight given by the kernel function. We will now formulate the rendering equation in the context of final gathering, where N_{fgr} final gather rays (FGRs) are shot via BRDF importance sampling. The radiance L_d contributed by diffuse indirect illumination along a primary ray in direction ω_o hitting a surface at point x with normal $\vec{n}$ is computed as:

$$L_d(x, \omega_o) = \frac{1}{N_{fgr}} \cdot \sum_{i=1}^{N_{fgr}} f_r(x, \omega_i, \omega_o) \cdot L_i(x, \omega_i) \cdot (\vec{n} \cdot \omega_i) \tag{6.1}$$

For normal photon mapping the radiance $L_i(x, \omega_i)$ along the i-th FGR is computed using density estimation from the k nearest neighboring photons found at maximum distance R_i from the hit point x_i

$$L_i(x_i, \omega_i) = \sum_{p=1}^{k} \mathcal{K}\left(\frac{\|x_i - x_p\|}{R_i}\right) \cdot f_r(x_i, \omega_p, \omega_i) \cdot \frac{\Delta\Phi_p(x_p, \omega_p)}{\pi \cdot R_i}, \tag{6.2}$$

where $\mathcal{K}(x_d)$ is the kernel function used in the density estimation. In *reverse photon mapping* the radiance $L_i(x_i, \omega_i)$ along a FGR is computed as follows:

$$\begin{aligned} L_i(x_i, \omega_i) &= \sum_{p=1}^{N_p} \mathcal{J}(x_i, x_p, R_p) \cdot f_r(x_p, \omega_p, \omega_i) \cdot \frac{\Delta\Phi_p(x_p, \omega_p)}{\pi \cdot R_p}, \\ \mathcal{J}(x_i, x_p, R_p) &= \begin{cases} \mathcal{K}\left(\frac{\|x_i - x_p\|}{R_p}\right), & \|x_i - x_p\| < R_p \\ 0 & \textit{else.} \end{cases} \end{aligned}$$

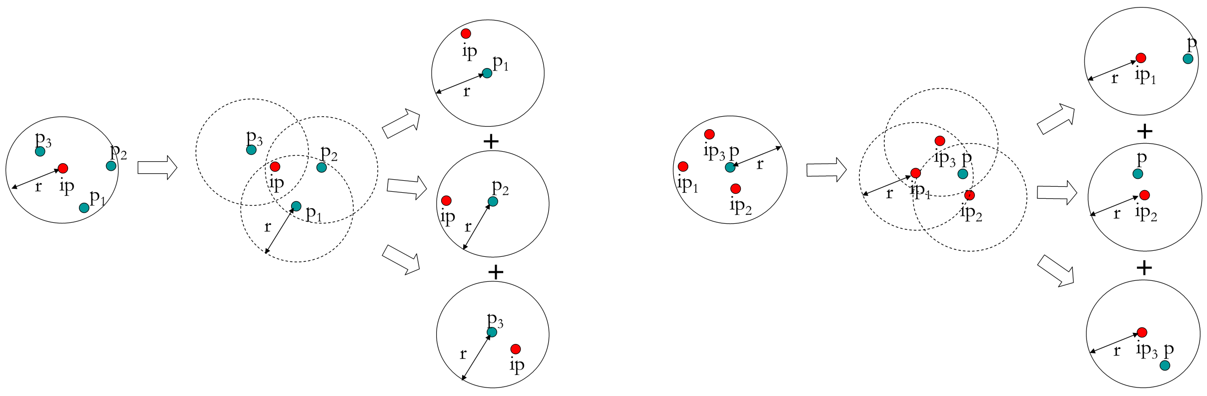


Figure 6.2: *Analogy between kernel density estimation (left) from the perspective of final gather rays (normal photon mapping) and (right) from the perspective of photons (reverse photon mapping). Point labels ip mark the final gather ray hits, p the photons and r the bandwidth or gather radius.*

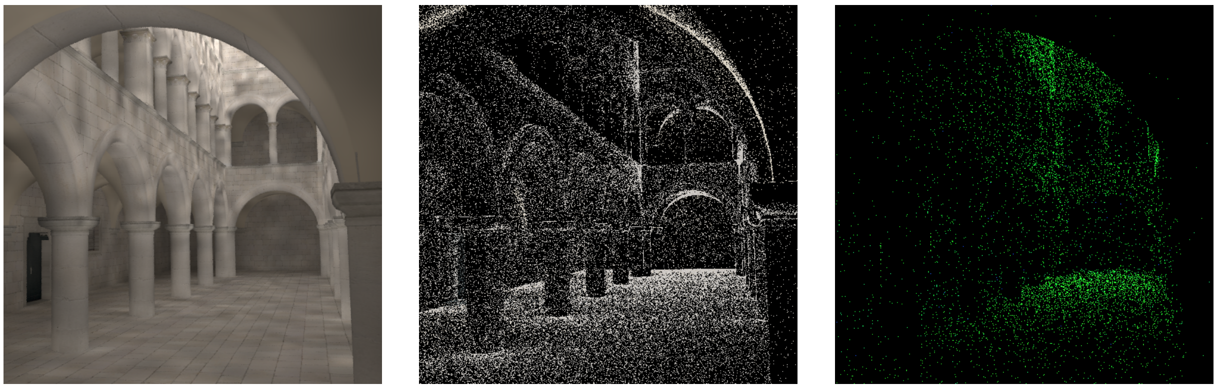


Figure 6.3: *Left: direct visualization of the irradiance in the Sponza scene computed from the photon map via reverse photon mapping. It shows what is seen by a final gather ray. Middle: a moderate distribution of the reverse photons (usually about 1000 times more), and right: the distribution of the photons (in false color).*

The interpretation is as follows: all photons p that found the reverse photon at position x_i with a maximum distance R_p from the photon position x_p, splat their flux $\Delta\Phi_p$ weighted by the kernel $\mathcal{K}$ to this reverse photon. Recall that the reverse photon is just a synonym for the end of the i-th final gather ray. The analogy between both density estimation techniques is depicted in Figure 6.2.

In order to allow for fast rendering using a direct visualization of the (adaptive) density estimation from the photon map, we must not initiate a final gather but store single reverse photons with directly visible pixel samples at primary ray hit points. Consequently, the resulting number of reverse photons is much smaller and the reverse photon map fits completely in the main memory. An example for the direct visualization of the density estimation computed with our method together with the distribution of photons and reverse photons, is shown in Figure 6.3. The blurry image on the left can be understood as what is seen by a final gather ray.

6.5 Estimating Kernel Bandwidth

In standard KDE the radius R (called bandwidth, support or window) of the kernel is globally constant and has to be adjusted to give satisfactory results. The setting for R is crucial (see Section 3.6). R is either tuned by the subjective choice of the user in a sense of trial and error or by applying optimization algorithms. The optimization is split in two groups: the parametric and the non-parametric estimation. The former assumes that the underlying distribution is of a particular class of analytical models (e.g. normal distribution, log-normal) and adjusts the parameters to compute the optimal bandwidth. Therefore parametric density estimation is not suited for photon mapping since we do not know anything about the distribution of the photons in advance. In order to provide some automatic adaptation to rather general distributions, several cross-validation approaches were developed such as *least squares cross-validation, biased cross-validation, likelihood cross-validation*. These methods are very expensive to compute and only feasible for a small number of samples. They were designed for statistical applications that often deal only with uni-variate (1D) data. Moreover, in the original form, these methods compute a constant optimal bandwidth as a trade-off between bias and variance. A globally constant bandwidth however works only well for low frequency distributions. When we are dealing with highly varying densities, it is recommended to use adaptive density estimation methods with varying bandwidth.

6.5.1 Precomputing K-Nearest Neighbors

So far we have been talking about KDE with constant bandwidth selection. Instead of using a fixed bandwidth R, we use an adaptive radius R_p where R_p is derived from the approximate density of photons similarly to normal photon mapping. This is superior to the "standard" kNN estimate as it does not suffer from long tails in the estimate as shown in Figure 6.4. The formula 6.3 shows the general form of *adaptive KDE* using different radii R_p for each photon p. We will first describe the *variable kernel method* that is based on the kNN method. For each photon p we search for the set Ω_p^k of the k nearest neighboring photons (kNN). We store the radius R_p which is the distance to the farthest photon in Ω_p^k in the photon p. During photon energy distribution, for each photon p we search for all reverse photons around p within the radius R_p. We splat the flux of p to these reverse photons as described in Section 6.4. This approach is independent of the reverse photon positions and hence, we do not necessarily obtain k nearest photons per reverse photon at least not in low density regions as shown in Figure 6.4. While the kNN estimate yields 500 nearest neighbor photons, for the variable KDE only 161 photons contribute to the reverse photons (red point). Moreover, the shape of the density estimation footprint is not bound to be circular but can have any shape as shown in the middle image of Figure 6.4.

The adaptive KDE does not only result in better density estimation but also in a faster and less complex search than for kNN density estimation. This is because we know the local gather radius R_p in advance and can use a constant bandwidth per photon during the search for the nearest reverse photons. Contrarily, the kNN density estimation usually requires maintaining a heap structure for finding the k-nearest neighborsin a maximum initial radius around the final gather ray hit point, which adapts to the density of photons during the search.

A problem with the adaptive KDE is that the radius R_p has to be precomputed for each photon. This can take long for a large number of photons. If the number of photons is N_p, we

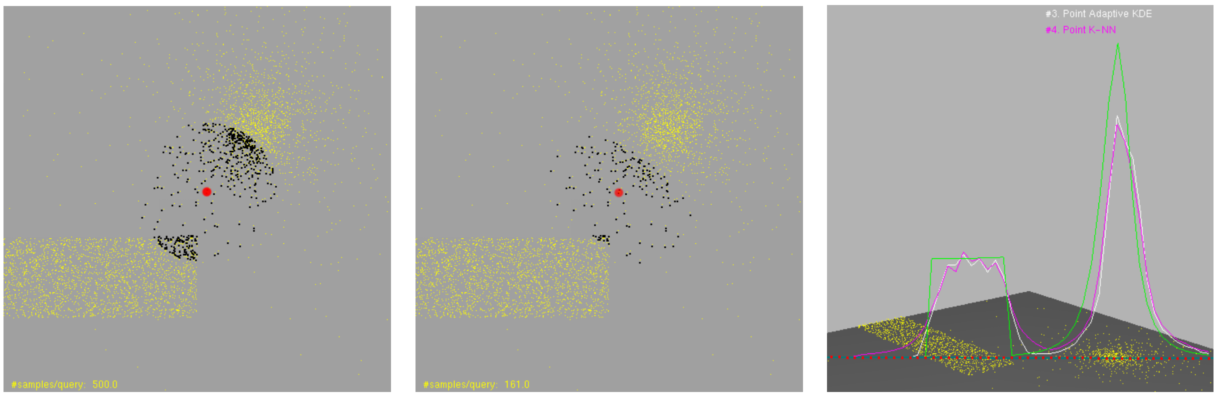


Figure 6.4: *The footprint of contributing photons (black points) to the irradiance estimate (red dot) for the kNN estimate (left) and for the variable KDE with precomputed k-nearest neighbors bandwidth per photon (middle). Both estimators use 500 (precomputed) nearest neighbors. The photon sample (yellow dots) is drawn from a density function consisting of a high frequency box function plus a simulating point light distribution function (peak with quadratic fall off). The right image shows the density estimated along the red dotted line using the kNN (purple curve) and the variable KDE (white curve) estimator. In this example both estimators use 100 (precomputed) nearest neighbors weighted by the Epanechnikov kernel. The green curve shows the reference computed analytically. For the visualization only a fraction of all photons is shown in the right image. Note that for the variable KDE in the middle image only 161 photons contribute to the irradiance estimate at the red dot, which reduces the tails in the low density region (that has almost zero density). It also preserves the gradients slightly better than the kNN estimate.*

need to perform N_p k-nearest neighbors queries to the photon map. This has a complexity of $\mathcal{O}(N_p \cdot (k + \log(N_p)))$. Still the radius precomputation for the k-nearest neighbor photons is faster than the kNN search for each density estimation point (reverse photon) if the number of reverse photons is larger than the number of photons. This is normally the case for final gathering.

Nevertheless, we can improve on the precomputation if we assume that the precomputed radius R_p does not change much among neighboring photons and is just an initial estimate of the local density. Moreover the exact k nearest neighbors per photon are not required. So, why not using a cheaper approximation to the bandwidth function?

We achieve this by discretizing the bandwidth function in space. In each leaf of the spatial *kd-tree over photons* we select m out of n photons and precompute the bandwidth R_i only at their locations. Next, we compute the average bandwidth $\bar{R}$ from the m radii R_i. All photons in the leaf get the same bandwidth $\bar{R}$. The concept works since the size of the leaves adapts to the photon densities and hence high-density regions get more bandwidth samples than low-density regions (see Figure 6.5). Using this simplification, the bandwidth can be efficiently precomputed for a large number of photons (only $10 - 20$ seconds for 500,000 photons and 16 photons per leaf) since the precomputation order is coherent in space and memory.

6.5.2 Estimating Bandwidth using the Kd-tree

In the previous section, we described a possible application of the well-known *variable kernel method* [74] to photon mapping. We observed that the precomputed kNN bandwidth changes relatively slowly compared to the photon density and that the spatial *kd-tree over photons* helps in reducing the number of kNN queries for the bandwidth precomputation. From the pilot kNN estimates, we computed the average bandwidth per leaf.

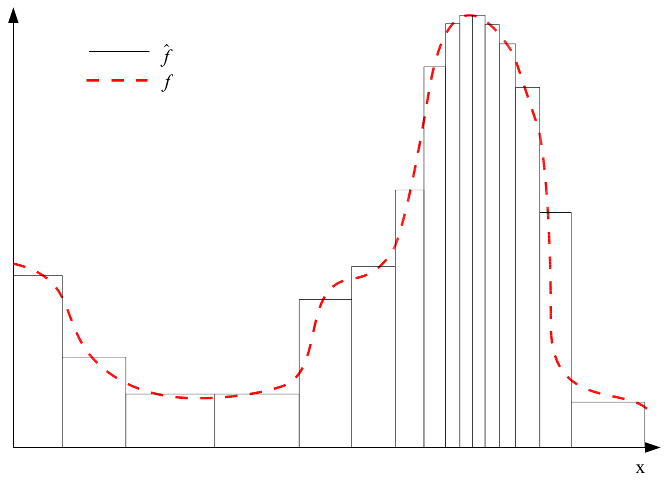


Figure 6.5: *The adaptive piece-wise constant density estimation precomputed in the leaves of the spatial kd-tree. The step function represents the leaves of the kd-tree in 1D, where the width along the x-axis corresponds to the size of the leaf's voxel and the height to its density. The error due to the piece-wise constant estimate decreases in the high-density region.*

The general view in the literature for statistics [1, 5] is that the adaptive kernel method is insensitive to the fine detail of the pilot estimate, and therefore any convenient estimate can be used [74].

Since the pilot estimate does not need to be accurate and the resulting density estimate is only visible through Monte Carlo sampling of a diffuse or glossy BRDF, we can even go one step further and avoid any kNN photon query by exploiting the spatial kd-tree. Recall that we store a certain number of photons per leaf whose corresponding voxel has size $a \cdot b \cdot c$. We compute the diagonal length $l = \sqrt{a^2 + b^2 + c^2}$. We use l and the number of photons in the leaf n to estimate the search-radius over a spatial cell associated with the corresponding leaf. The upper bound for the bandwidth R that aims to finds the k nearest photons is then computed as follows:

$$\begin{aligned} \tilde{R} &= \sqrt{(\frac{1}{2} \cdot l)^2 \cdot \frac{n}{k}} \\ R &= \max(R_{min}, \min(\tilde{R}, R_{max}), \end{aligned}$$

where k is the number of photons in the actual kNN search, R_{min} and R_{max} are constant lower and upper bound respectively. They can either be determined by the size of the scene or by a user defined smoothness parameter. We compress R to 1 Byte and store it in the photon record which we need to decompress before the actual search in the reverse photon map. We get a pessimistic estimate for R since the surface area for a leaf is over-estimated (by using the diagonal, we consider the bounding sphere of the leaf). Hence the initial pilot density is underestimated and R is larger. Nevertheless, this method can provide us approximately the desired number of photons (k) and hence the local density when searching that region (Figure 6.5). Since the photon density for diffuse illumination is changing slowly the photon density gradient for neighboring leaves is small. Note that the size of a leaf adapts to the density but the number of photons per leaf stays approximately the same. Figure 6.6 shows the density estimation visualized in the chart on the right from the kd-tree leaves for the test scene on its

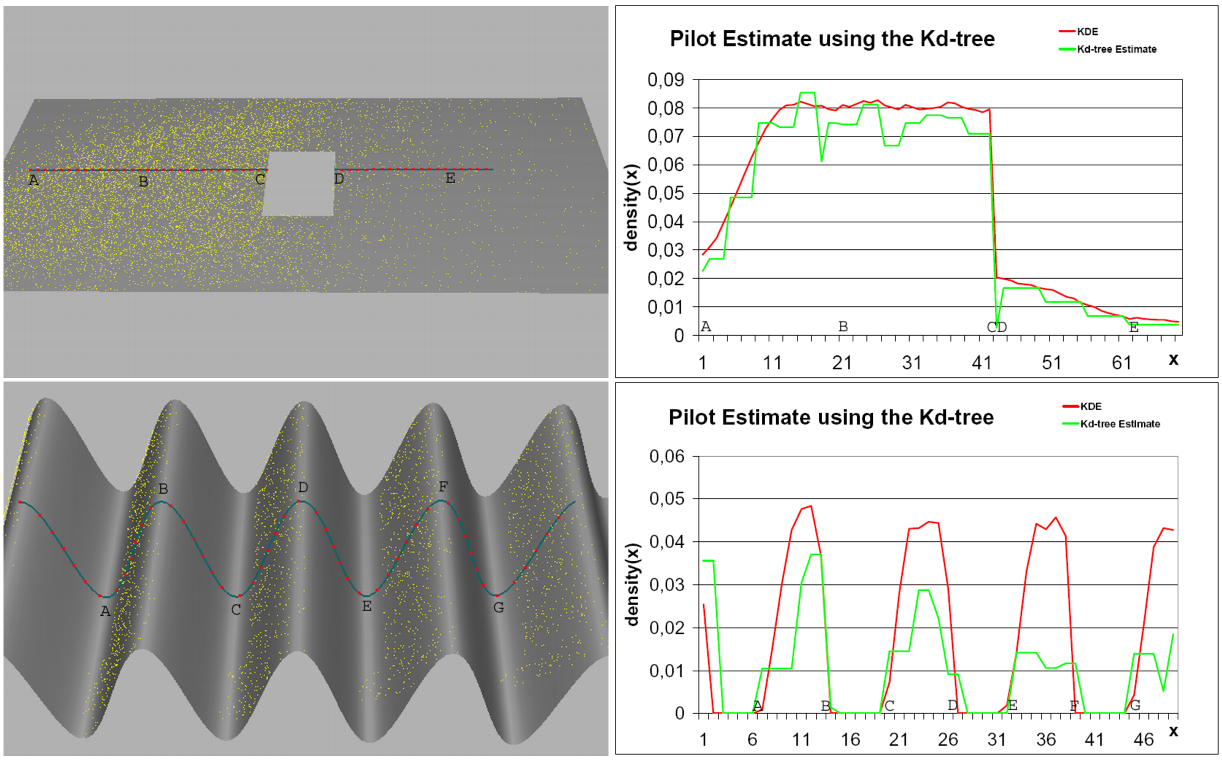


Figure 6.6: *The images on the left show the test scenes and the photon distributions. The chart on the right shows the density $f(x)$ along the depicted line on the left. The density is estimated with the KDE method (red line) and the proposed kd-tree method (green line). One can observe, how the green step function corresponds to the density in the leaves of the kd-tree.*

left. Figure 6.7 shows the photons bandwidth in a false color image where blue corresponds to the largest bandwidth and red to the smallest (approx. 3 cm gather radius).

6.5.2.1 Some Implementation Issues and Optimizations

So far we used the approximative density estimation from the leaves of the kd-tree as the pilot estimate. We experienced that the number of photons n used to compute the density for the pilot estimate should be large. This brings up two problems. First of all, we only use $n = 32$ photons per leaf. Hence, the pilot estimate is too noisy to give robust results. And second, n is independent of the total number of photons. We could use more photons per leaf but this would decrease the spatial coherence of photons since the leaves would become larger and photons inside are unsorted. This is not recommendable since it may slow down the kNN query performance used for the density control (Section 6.9.3) and makes the *aggregate search* (Section 6.9.2) sub-optimal. Therefore, we decided not to estimate the density from the leaves but from a sub-tree that is a few levels up in the hierarchy. To do so, we precompute the bandwidth whenever a sub-tree node contains less than the desired number of photons (≈ 100) and assign this bandwidth to all photons in the sub-tree. We also reduced the maximum number of photons per leaf to 16.

Since a larger part of the tree uses now the same bandwidth, it is wasteful to store the bandwidth with each photon record. Hence, we move the bandwidth stored in the photons to a separate array A_l associated with the sub-trees. Array A_l contains the bandwidth R and the offset to the next sub-tree which is the current number of photons in the sub-tree. Since we access the photon array sequentially during photon splatting, we can simply look-up the gather

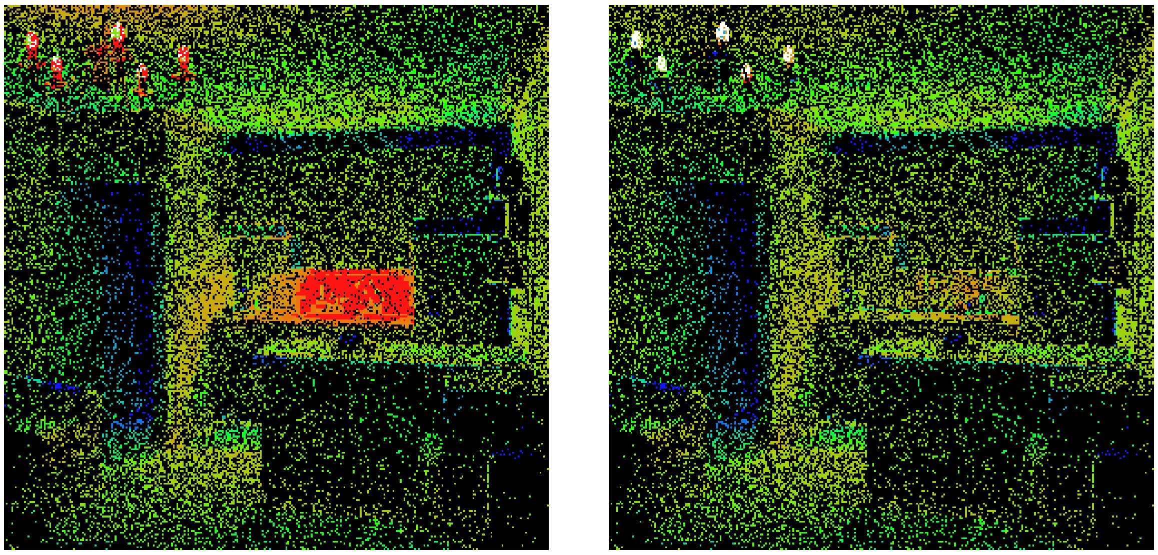


Figure 6.7: *Visualization of the precomputed bandwidth per photon in false colors (from blue ≡ largest to red ≡ smallest gather radius). Left image shows the photon map without density reduction and the right image shows the photon map after density control was applied. Note the change in the bandwidth after density reduction.*

radius R in A_l and increment A_l by its offset during traversal.

Furthermore, we use only the squared gather radius R^2 throughout the whole algorithm. This eliminates the computation of the square root in the radius precomputation phase and during kd-tree searches including photon splatting.

6.6 Run-time complexity

In normal photon mapping with final gathering the recommended number of final gather rays (FGR) per pixel is between 600 and 4255 [9], [40], [78]. The number of pixels in an image is typically 1 - 3 millions. This gives in total 10^9 to 10^{10} FGRs to be computed. It involves creation of FGRs via BRDF importance sampling, tracing the rays through the scene, and estimating radiance at their hit points with the scene. Let us denote by G the number of final gather rays per pixel, by I the number of pixels in the image, by K the number of photons searched in the neighborhood, and by R the radius used by reverse photon mapping for fixed-radius (i.e. constant bandwidth) search. For the analysis, we assume the ideal kNN search finding always K nearest photons in $\mathcal{O}(K + \log_2 N_p)$ time.

6.6.1 Normal Photon Mapping

For normal photon mapping (NPM), we shoot N_p photons, we construct the kd-tree in the time $T^C_{NPM} = c_1 \cdot N_p \cdot \log_2 N_p$. During rendering, we shoot $N_{fgr} = G \cdot I$ FGRs. For every FGR, we find and process the K nearest photons in time $T^S_{NPM} = c_2 \cdot K + c_3 \cdot \log_2 N_p$. The total number of found photons for all FGRs is $N_{fgr} \cdot K$. The total computation time for normal photon mapping without construction time is then:

$$T^T_{NPM} = N_{fgr} \cdot T^S_{NPM} = c_2 \cdot N_{fgr} \cdot K + c_3 \cdot N_{fgr} \cdot \log_2 N_p \tag{6.3}$$

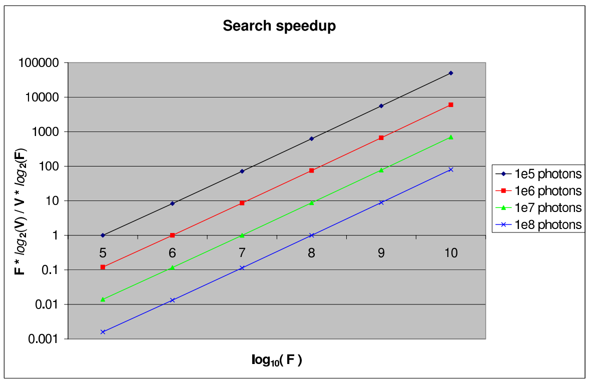
The constant c_1 is a factor behind the sorting of the photons during the construction of the tree, c_2 is the factor for processing a single pair photon×FGR (i.e. feasibility test and evaluation of the kernel function), and c_3 is the time required for a single traversal step in the kd-tree during kNN search. These constants are determined by the actual implementation and are therefore not relevant for the theoretical analysis of the algorithms. They can be excluded for the time complexity analysis, though they should not be neglected. For example, a recursive implementation of the kd-tree search is about 10 percent slower than its iterative counterpart. The space needed for the photon mapping is $\mathcal{O}(N_p)$.

6.6.2 Reverse Photon Mapping

For reverse photon mapping (RPM) we construct two kd-trees: the kd-tree over N_p photons (photon map) and the kd-tree over N_{fgr} reverse photons (reverse photon map). Since we increase the number of photons in a single leaf to T_m, the construction time is $T^C_{RPM} = c_1 \cdot (N_p \cdot \log_2 \lceil N_p/T_m \rceil + N_{fgr} \cdot \log_2 \lceil N_{fgr}/T_m \rceil)$. The searches are carried out in the reverse photon map. Let us assume that the total number of found pairs *photon* $\times$ *reverse photon* is also $N_{fgr} \cdot K$. Since the number of searches is N_p, the number of reverse photons found per search is on average $K_r = N_{fgr} \cdot K/N_p$. The search time in the reverse photon map for a single photon is then $T^S_{RPM} = c_2 \cdot K_r + c_3 \cdot \log_2 N_{fgr}$. The total computation time neglecting construction time is then:

$$\begin{aligned} T^T_{RPM} &= N_p \cdot T^S_{RPM} \\ &= c_2 \cdot N_p \cdot (N_{fgr} \cdot K/N_p) + c_3 \cdot N_p \cdot \log_2 N_{fgr} \\ &= c_2 \cdot N_{fgr} \cdot K + c_3 \cdot N_p \cdot \log_2 N_{fgr} \end{aligned} \tag{6.4}$$

The space complexity for reverse photon mapping is $\mathcal{O}(N_p + N_{fgr})$. From the formula 6.4 we can already see that the computation time is the same for NPM and RPM if the number of photons N_p and the number of final gather rays N_{fgr} is equal. In our analysis the computation time to process all pairs *photon* $\times$ *reverse photon* ($T^P = c_2 \cdot N_{fgr} \cdot K$) is the same for both normal and reverse photon maps ($c_2 \cdot N_{fgr} \cdot K$ in formulae 6.3 and 6.4). The processing time T^P includes the time for computing the radiance contribution per FGR via BRDF evaluation. However, the searching time is significantly smaller for RPM. The theoretical speedup for RPM, excluding T^P, is shown in Figure 6.8 on the left. The ratio of computation times T^T_{NPM}/T^T_{RPM} is shown in Figure 6.8 on the right for a various number of FGRs N_{fgr} and different numbers of photons N_p. From the analysis it follows that the reverse photon mapping algorithm decreases the searching time only if the number of final gather rays to process is by orders of magnitude higher than the number of photons. This is typically the case for rendering images even when using irradiance caching, in particular high resolution images used in the production rendering. It should be mentioned that if the computation time T^P dominates the computation, reverse photon mapping hardly gives any speedup. The dependence of RPM on the number of photons is summarized in Table 6.1. Each test uses the same setting for the eye pass (i.e. same number of reverse photons) but approximately the double number of photons. For testing purposes the density control (Section 6.9.3) was switched off and the adaptive bandwidth (Section 6.5.2) was bound to a maximum gather radius of 1 meter. If the adaptive bandwidth is not bounded by a maximum radius, the timings become similar for a large range of photon numbers. This



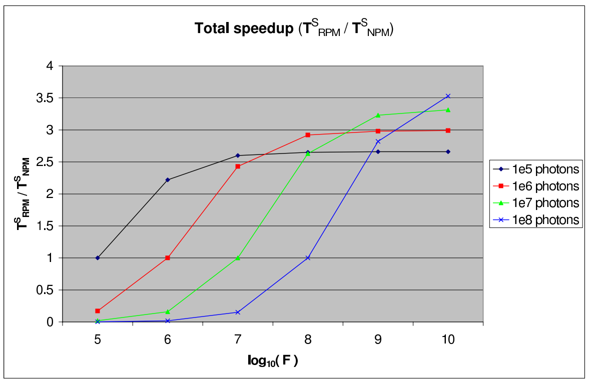


Figure 6.8: *The theoretical speedup for reverse photon mapping compared to normal photon mapping. The number of pairs photon×reverse photon (i.e. all photons found by all FGRs in normal photon mapping) is the same. The constants are $K = 100$, $c_2 = 0.1$, and $c_3 = 1.0$. (left) The theoretical speedup only for the search. (right) The total speedup comprising search and processing of the pairs photon×reverse photon.*

# Tasks	N^S_{ep}	N_p	$T^T_{RPM}[s]$	$T^S_{RPM}[s]$
97	$2.00 \cdot 10^9$	12,670	$3,793$	297
97	$3.75 \cdot 10^9$	24,701	$3,872$	308
98	$3.52 \cdot 10^9$	50,093	$3,778$	297
99	$5.98 \cdot 10^9$	99,383	$4,073$	465
99	$10.80 \cdot 10^9$	199,705	$4,312$	826
100	$21.00 \cdot 10^9$	400,178	$5,063$	$1,522$
102	$38.90 \cdot 10^9$	800,025	$6,344$	$2,788$
104	$67.60 \cdot 10^9$	1,602,241	$8,417$	$4,722$

Table 6.1: *Rendering timings for the atrium scene using reverse photon mapping with different numbers of photons (without density control). The adaptive bandwidth is used with rendering setting: 800 final gather rays per sample, 4 samples per pixel, and a resolution of 500×500 pixels (i.e. $\approx 800 \cdot 10^6$ reverse photons). From the left column to the right column: the number of tasks generated by the scheduler, the total number of gathered reverse photons N^S_{ep} from all tasks and all queries, the number of stored photons N_p, the total computation time T^T_{RPM} in seconds (see Table 6.5 for a complete list of timings of all rendering stages), and only the time for rendering the indirect diffuse illumination via density estimation T^S_{RPM} including search of reverse photons. Note that the number of rendering tasks increases slightly with the number of stored photons N_p. This is enforced by the given memory limits in the scheduler described in Section 6.7.*

is because the number of gathered reverse photons per photon increases automatically when the number of photons decreases. Hence the total number of found reverse photons to be processed is similar for different numbers of photons. This also shows the weakness of kNN density estimation: The large bandwidth in low density regions leads to blur and light leakage. Moreover, more cache misses are produced if a photon query includes too many nearest neighbors which can even result in a decrease of performance if the number of photons is reduced!

6.7 Scheduling Tasks

Rendering images in high resolution and high quality requires a large number of reverse photons (i.e. FGRs) that do not fit in the physical memory of consumer-level computers. Nowadays, the main memory of most standard PCs (with 32 Bit addressing) is limited to 1–4 GBytes. In our current implementation, we allow for up to $2^{30} \approx 1$ Billion entries in the reverse photon array. The storage space required is then $2^{30} \times 32$ Bytes = 32 GBytes, which is far from affordable for the main memory of a consumer-level computer nowadays. (Besides it is even infeasible for most computers with a 32 Bit address space.) One way to solve this issue is to subdivide the data execution in many tasks and combine the results of these tasks.

The major part is the computation of the radiance along the final gather rays. Since the computation of density estimation via reverse photon mapping does not induce any correlation between FGRs nor any ordering of the pixels, we can subdivide the computation in the spatial as well as the "temporal" domain. The former is carried out by tiling the whole image plane and the latter by processing the pixels in several repeating runs splitting the number of FGRs per pixel.

In order to make it practical to the user, we hide the burden for setting these parameters manually. Therefore, we implemented a memory scheduler that does the job of computing the number of tiles and the number of final gather rays for the tasks in advance. The input for the scheduler is the size of available memory for the renderer, the image resolution, the number of samples per pixel, the number of FGRs per pixel sample, the number of caustic and global photons to store, and the user preferences as a trade-off between image quality and the tile size. The latter produces faster results however with lower quality and vice versa.

The scheduler creates the sequence of tasks to be computed by the renderer. A task is specified by the image tile coordinates, the number of samples per pixel, the number of FGRs per pixel sample, the weight for this tile (i.e. its contribution to the overall image), the initial random seed, the estimated memory consumption by reverse photons and photons for this task, and some flags specific to the rendering and statistics (e.g. whether we want to re-shoot the photons or which camera-frame-sampling scheme we want to use). When a tile is computed, all the computed tiles are combined to a single image.

The order of tasks can also allow progressive previewing: first the whole image is computed with a small number of FGRs per pixel, and then the quality is improved by computing the tiles, increasing the number of FGRs per pixel. The proposed scheduling is also a natural candidate for parallelization. Note that the photons from the light sources are only generated for the first task and reused for all subsequent tasks. This is not obligatory. We could delete the photon array and re-shoot them using a new seed for successive runs. Using Quasi-Monte Carlo sampling (e.g. Halton sequence), we would be able to obtain the same results in successive runs as if using a multiple of the number of photons per run at once. This way we could save memory

```
int spp  = samplesPerPixel;
int fgrs = finalGatherRaysPerSample;
int width  = imageWidth;
int height = imageHeight;
int tilesY = 1;
int tilesX = 1;
int sampleWeight = sampleWeightIncrement;
int tileWeight   = tileWeightIncrement;
estimateMemoryConsumption( & memoryAvailable, & memoryNeeded,
                           width, height, spp, fgrs);
while (memoryAvailable < memoryNeeded) {
    if (sampleWeight > tileWeight) {
        if (height > width) {
            tilesY++;
            height =  (int)ceilf( imageHeight /  (float)tilesY);
        }
        else {
            tilesX++;
            width =  (int)ceilf( imageWidth / (float)tilesX);
        }
        tileWeight += tileWeightIncrement;
    }
    else {  // decrease the number of samples per pixel
        if (spp > 1)
            spp = spp / 2;
        sampleWeight += sampleWeightIncrement;
    }
    estimateMemoryConsumption( & memoryAvailable, & memoryNeeded,
                               width, height, spp, fgrs );
}
nTasks = tilesX * tilesY * (int)ceilf(samplesPerPixel / (float)spp);
```

Figure 6.9: *A basic scheduler for reverse photon mapping*

by using less photons per run which in turn could be used for storing more reverse photons. However, the photon distribution should be the same for all tiles of the image. Otherwise, visible artifacts between neighboring tiles may occur. To our experience the subjective image quality of the global illumination with final gathering is not much affected when using more photons but depends heavily on the number of final gather rays per pixel. However, we have not experimented with the influence of using more photons in the final gathering.

6.7.1 A Basic Scheduler for Reverse Photon Mapping

We have implemented two scheduling algorithms. The first one is the simpler algorithm that computes a pessimistic estimate for the maximum number of reverse photons per tile and sub-divides the image and the number of samples per pixel until the needed memory per task is less than the available memory. The algorithm for the task sub-division is shown in Figure 6.9. The function `estimateMemoryConsumption` computes an estimate of the memory used by the photons and kd-tree over photons, the images, the scene geometry including final gather ray cache and write-back cache for reverse photons and returns the remaining memory in variable `memoryAvailable`. Additionally, it computes the memory needed to store the reverse photons for a given tile with parameters `width, height, spp, fgrs` and returns it in `memoryNeeded`.

Next, the image is sub-divided into the number of tiles in x and y direction. Tile coordinates and samples per pixel for each tile are written to the task list. At the end the weights for each tile are computed from the ratio of the actual samples per pixel used and the final number of samples per pixel. Note, that a user-defined preference parameter determines the weights `sampleWeight` and `tileWeight` that dictate the partitioning into the number of repeated runs per pixel and the number of image tiles, i.e. two times a higher sample weight results in two times more tile sub-divisions than pixel sample splits.

6.7.2 Scheduling with Improved Memory Estimation

The basic scheduler explained previously has a major drawback. It cannot accurately predict the memory consumption by the reverse photons since it depends on the scene and the involved BRDF sampling of final gather rays. Hence, the number of stored reverse photons varies highly for different scenes using the same rendering settings. Moreover, secondary final gather increases the number of final gather rays in corners considerably. In the former scheduler, we simply assumed the number of stored reverse photons is maximum in each tile. This often leaded to large over-estimations for scenes with mostly glossy light transport and did not work at all when secondary final gather was enabled. In order to minimize the number of tasks, which is beneficial for reverse photon mapping since it can exploit more memory, we precede an initial pre-sampling step. This pre-sampling over the entire image plane identically imitates the final rendering process including final gathering and BRDF sampling. However, it uses a much smaller number of samples per pixel. And instead of storing the reverse photons at final gather ray hit points, we increment the number of stored samples in a matrix field which corresponds to the currently sampled pixel of the image. The matrix has exactly the same size as the image and is initialized to zero at the beginning.

Once we have pre-sampled the whole image and filled the matrix, we build a summed-area table from this matrix. Using this table, we can easily compute the memory that would have been used by reverse photons during pre-sampling in a rectangular area of the image if we had stored them. Since we know the ratio of initial samples and final number of samples for the image, we can compute a precise estimate for the required storage space of each area in the image.

We pass the summed-area table to the scheduler which adaptively constructs the tiles in a way that each tile uses approximately the same memory and the memory consumption per tile is maximized. The tiles are not regularly sized anymore but adapt to the local sampling distribution. For example, glossy image parts will receive less samples and the tile size will automatically increase in this region. This adaptation would be also beneficial for parallel computation of the tasks since the workload and hence the computation time is balanced among concurrent tasks.

So far, we have neglected a small detail in the algorithm. We have described the adaptive scheduler for standard final gathering via BRDF importance sampling. However, we also allow the user to enable secondary final gathering in order to obtain slightly better results in corners. If we allow for secondary final gathering, the pre-sampling results are not linear scalable anymore since the results grow exponentially in the corners. We do not know the pixels affected by secondary final gathering and additionally the number of secondary final gather rays (FGR^{2nd}) depends also on the number of primary final gather rays. Therefore, we keep it simple and store all FGR^{2nd} in a second matrix. At the end the final estimate for the number of reverse photons per tile is computed as the sum from both matrices (i.e. summed area tables) that are scaled individually.

6.7.3 Irradiance Caching

Irradiance caching is a widely used technique for accelerating rendering of indirect diffuse illumination by interpolation and works smoothly hand in hand with photon mapping. It normally yields a speedup compared to normal photon mapping (NPM) of one order of magnitude. This is a better speedup than we achieve with reverse photon mapping. Below we describe how irradiance caching can be combined with reverse photon mapping. We will not explain the details here since we have already introduced the general concept in Section 4.5.1 on page 50.

The difficulty in reverse photon mapping (RPM) is that we do not know the irradiance during final gathering. So, we cannot immediately interpolate the irradiance from neighboring cache samples at the query location and add its contribution to the image as it is done for normal photon mapping. In RPM the radiance per FGR is computed during photon splatting and we get the irradiance only at the end. Hence, we need to store the irradiance samples that are to be interpolated and process them in a second pass (completion phase). Fortunately, we already store all pixel samples (primary ray shot through the image plane) for pixel filtering purposes (convolution across the sub-pixels). Thus, we simply set the state of the pixel sample to interpolation. For that we do not need to introduce an additional variable.

At the end, after the cache is complete and all irradiance values for each cache sample have been computed in the processing phase, we can start the second pass, the actual interpolation. To do so, we go over all pixel samples that are marked for interpolation and query the irradiance cache for neighboring cache samples in the same way as it is done for NPM described in Section 4.5.1. However, this time we benefit from the complete irradiance cache and do not need to use progressive refinement for the image sampling as for the standard irradiance cache. This second pass algorithm even yields a cleaner and smoother interpolation which can be seen in Figure 4.7 on page 55. This is because each query to the irradiance cache returns *all* re-usable neighbors, their correct weights, and their irradiance. We simply sum all weighted irradiance values, divide the result by the sum of weights, and write this result back to the interpolated pixel sample. Now, we have got the irradiance at the sample location. To compute the final radiance for the pixel sample, we multiply the interpolated irradiance by the constant BRDF vector (diffuse albedo divided by π) stored with the pixel sample.

The second pass, the irradiance cache completion, is done in negligible time compared to the rest of the computation. The overhead for querying the cache twice is negligible due to the coherent sequential processing. However, the increased memory requirements are considerable since we need to store the whole irradiance cache and the pixel samples for a tile containing position, compressed normal, and albedo at primary ray hit points. Moreover, irradiance caching cannot be separated into tile-based rendering passes since we also need to query cache samples from previous tiles in order to interpolate the irradiance smoothly across the tile boundaries. Splitting the irradiance cache to tiles yields discontinuities at tile boundaries. Therefore, the complete cache must be maintained and updated during the task processing loop.

The quality of the rendered images with irradiance caching depends strongly on the user defined error threshold and the number of neighboring cache samples for the interpolation. Below are the statistics for three rendering settings for the sponza scene. The results are shown in Figure 6.10.

Error	N_{fgr}	N_{ep}^T	N_Q/N_Q^* [%]	$\bar{N}_K$	$\max N_K$	T_{fgr} [s]	T_{de} [s]	T_{total} [s]
0.2	400×5	$3.4 \cdot 10^6$	99.45	5	17	21.4	23.0	124.9
0.1	400×5	$15.1 \cdot 10^6$	97.54	4	10	53.5	95.5	235.8
0.0	400×5	$614.4 \cdot 10^6$	–	–	–	1,910.4	2,431.0	4,826.7

Table 6.2: *Irradiance cache and rendering statistics for RPM for the sponza scene obtained from 2 different error settings and from the reference with disabled irradiance caching. All images were rendered in a resolution of* 640×480 *pixels with 5 super samples per pixel using 700,000 photons for density estimation. From left to right column: the maximum error (a) for cache interpolation, the number of final gather rays per pixel (N_{fgr}), the total number of stored reverse photons ($N_{ep}^T \equiv$ number of final gather rays), the percentage of interpolated irradiance values (N_Q/N_Q^*), the average ($\bar{N}_K$) and maximum ($\max N_K$) number of computed irradiance values used for interpolation at one point, the time spent for final gathering and irradiance caching T_{fgr}, the time for computing the indirect illumination via density estimation from 700,000 photons and 50 nearest neighbors (T_{de}), and the total time (T_{total}) spent for the whole rendering which includes photon tracing (63 s), photon map pre-processing (10 s), tree construction, and post-processing (filtering) (6 s).*

6.8 Algorithm Workflow

Here, we will describe the algorithm flow for a single task given by the scheduler. The overall scheme of the processing is depicted in Figure 6.11.

The algorithm decomposes the computation to several tasks to deal with the increased memory requirements. In the initialization the data structures are allocated and initialized using the parameters that are read from the global environment. The pre-sampling of the image plane is performed and the scheduler is called which returns the set of tasks for rendering. Each task undergoes the whole rendering pipeline from initialization of the camera to image tile composition. The camera initialization includes image tile and samples per pixel setting. Next the image plane is sampled and the direct light is rendered at primary ray hit points (pixel samples). At each diffuse and moderately glossy hit point, final gathering is carried out accounting for irradiance cache queries and secondary final gather. For each final gather ray hit point an reverse photon is stored. At the end the reverse photon map is constructed.

The processing phase starts with photon shooting and global photon map and caustic photon map construction. Both maps are individually filled in a few iterations until we get approximately the required number of photons per map. However, the caustic map filling is accelerated using a kind of projection map that is generated during the initial photon shooting and used after one fifth of the required number of photons has been stored. The projection map is a boolean matrix where each element corresponds to a certain direction that is either marked as a caustic path or not. If a maximum number of emitted photons is exceeded during the caustic photon map filling or not a single caustic photon has been stored after 100.000 traced photons, the shooting phase is always aborted. The construction of the photon maps follows optionally the density control which reduces the density to a user-defined threshold and hence decreases the number of photons. Next, the density estimation is computed in the reverse order. The photons from the global photon array A_p are processed sequentially discarding the kd-tree over photons, which creates a highly coherent search pattern in the reverse photon map. For a processed photon the neighboring reverse photons are gathered and the photon energy is distributed to

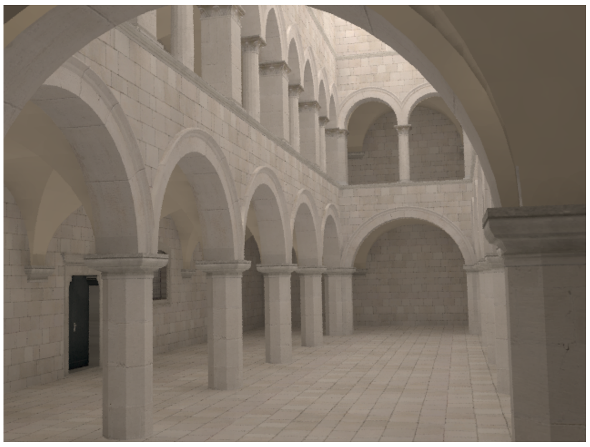
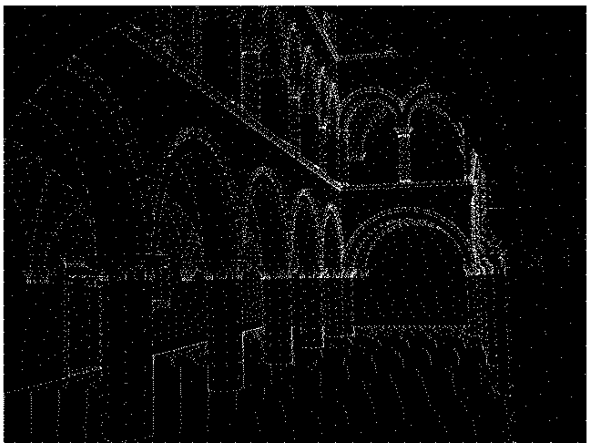


Image (a) 129 s


$error = 0.2$

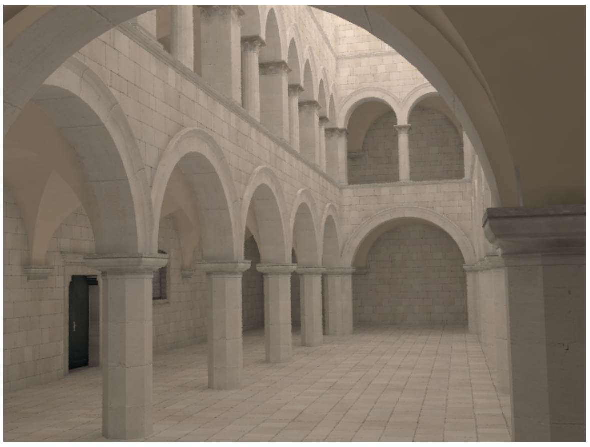
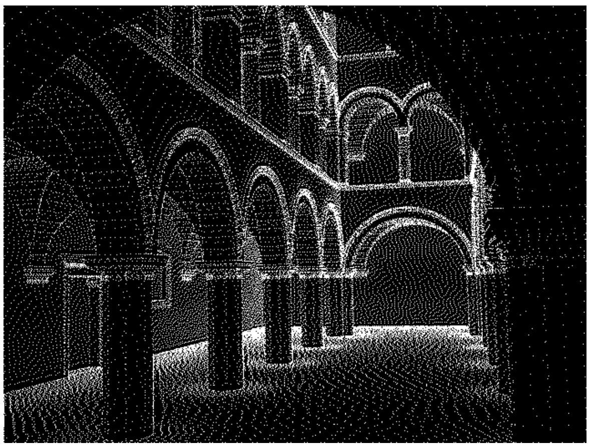


Image (b) 236 s


$error = 0.1$

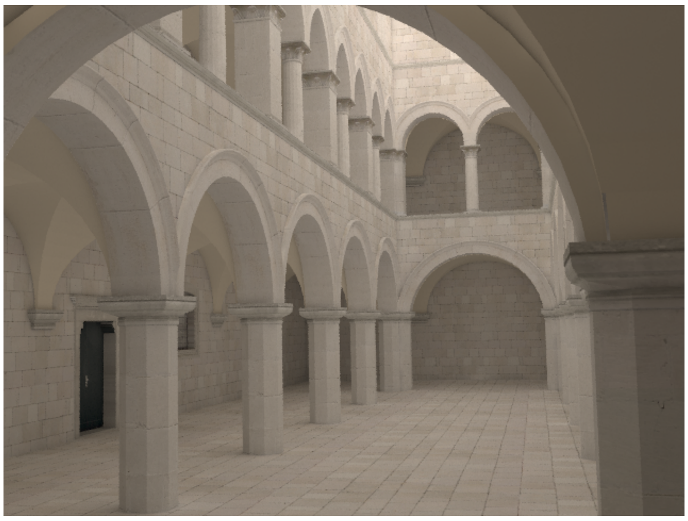

Reference: 4827 s

Figure 6.10: *Irradiance caching with reverse photon mapping for the sponza scene. All images were computed using* 400×5 *final gather rays per pixel in a resolution of* 640×480 *pixels. Whereas for Image (a) (top) a low quality setting with a large error bound was used, for Image (b) (middle) a smaller error was set (see Table 6.2). The bottom image shows the reference computed by RPM using the same setting but without irradiance caching.*

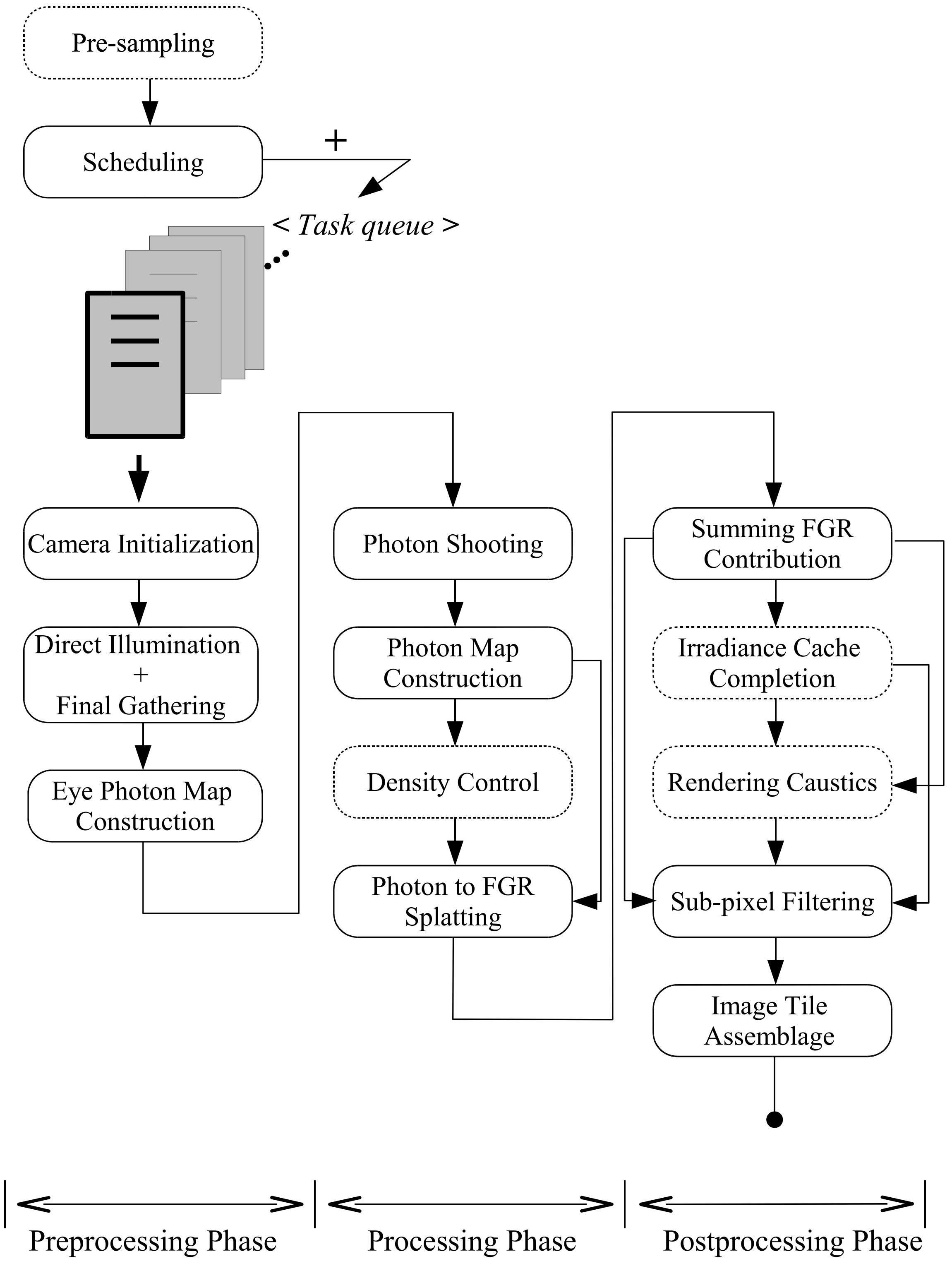


Figure 6.11: *Algorithm pipeline of reverse photon mapping. The dotted round boxes represent optional steps. The work flow is divided up to three main phases (Preprocessing, Processing, Postprocessing) that are executed for each task.*

the corresponding entries in the write-back cache array A_w using a supported filter kernel (e.g. Box, Cone, Biweight). The entries of A_w map one to one to the entries of the reverse photon array A_r. Note, that during the search the photon array A_p and the reverse photon map are used for read-only access. Only A_w is read to the CPU cache and written back to the main memory (see Section 6.9.4.1 for details).

In the postprocessing phase, the accumulated radiance in $A_w[i]$ per reverse photon $A_r[i]$ is weighted by the reverse photon's contribution given by BRDF importance sampling of FGRs and added to the array of pixel samples A_s at index given by the reverse photon. The array A_w is processed in sequential order. Next the irradiance cache is completed. The irradiance cache entries are searched by all pixel samples that are marked for interpolation and their irradiance is extrapolated to the pixel samples as in the normal irradiance cache algorithm. The irradiance cache is optional and can be controlled by the user. In the next step the caustic contribution is computed for each pixel sample and added to its pixel radiance. The caustics are computed as described in [34] using standard kNN density estimation from the caustics photon map. Again, this step is optional and can be switched on and off by the user. After that, the image (tile) is formed by adding the radiance along pixel samples to the pixels possibly filtered using a convolution kernel (Box, Cone, Gaussian, Mitchell [55]). Finally, the complete image is updated by the current image (tile) that is multiplied by the weight of the task. The data flow between data structures for the density estimation including searching and pixel reconstruction is visualized in more detail in Figure 6.12.

There are several algorithmic design goals fulfilled by our algorithm. First, the computation over the data structures results in highly coherent access to the main memory. Due to the read-only access to reverse photons by using the write-back cache for accumulating radiance, the write-back traffic from the CPU cache to the main memory is efficiently reduced. Third, the increased main memory requirements are alleviated by efficient methods using tiling, irradiance caching, and external caching on hard disk.

6.9 Optimizations

In this section we will cover the important algorithmic optimization techniques that we have applied and tested with our implementation.

6.9.1 Ray Shooting Cache for Final Gather Rays

Ray tracing of final gather rays (FGRs) is carried out for subsequently created FGRs from BRDF sampling on visible surfaces. Since their generation is basically a random process (Monte Carlo sampling), the directions of subsequently created FGRs are incoherent. Test for visibility data structures have shown that coherent ray tracing can be up to three times faster than highly incoherent ray tracing [63]. Here, we propose an organization scheme to improve on coherency for final gather ray tracing (referred to as final gathering). We create a single cache for FGRs by subdividing a sphere surface to N_C directional cells C_i covering a certain solid angle. Each C_i is associated with an array to store N_R FGRs to be ray traced. We set $N_R = 100$. Successively, we store the created FGRs in the cache by mapping the direction $\vec{D_o} \in R^3$ to a cell C_i identified by $(\theta, \phi) \in N^2$ (see Figure 6.13). Whenever the array of a cell is full, we ray trace all the FGRs in the array and store reverse photons at their intersection points with diffuse or glossy surfaces. One difficulty in this approach is the application of secondary final gather in corners

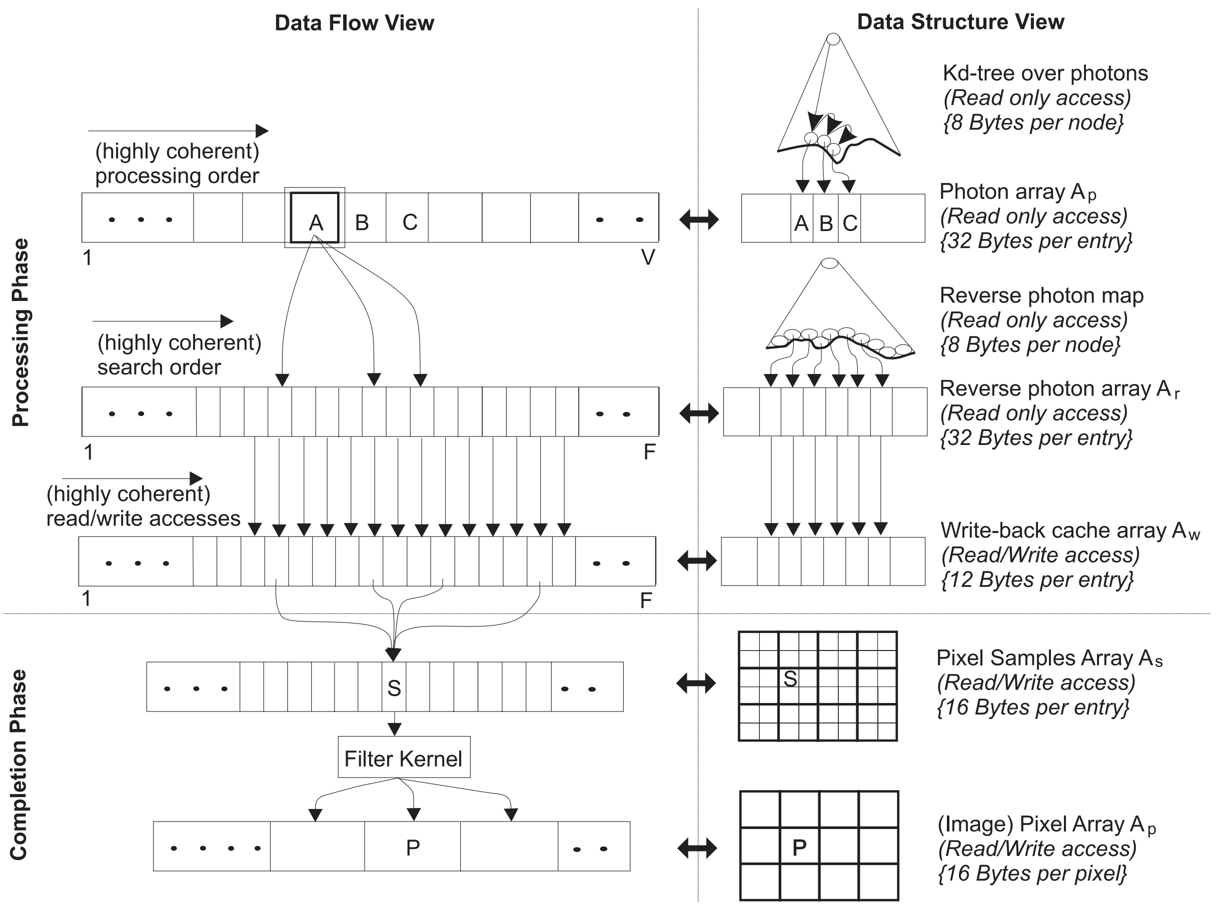


Figure 6.12: *Visualization of the data flow on the left and the data structures on the right. In the processing phase (top) the radiance for all FGRs is computed by density estimation (i.e. splatting the energy of each photon to neighboring reverse photons). In the completion phase (bottom) the contribution to the pixels is computed, possibly using a filter kernel. Note that the majority of data accesses is read only and highly memory coherent.*

and on specular surfaces. Due to the discretization of the sphere, it can happen that we flush a particular strata whose rays travel a very short distance or hit a specular surface in a grazing angle. Then a secondary final gather (only one ray in case of a specular hit) might be invoked. Occasionally, a secondary final gather ray is added to the same strata as its primary final gather ray (see Figure 6.14). Even if this case happens rarely, we need to catch it in order to guarantee robustness.

The final gather ray cache yields up to 30 percent speedup in ray shooting for complex scenes. For simple scenes consisting only of a few primitives such as the cornell box scene, it can perform worse than the naive final gathering. This is due to the overhead for organizing the FGR cache which becomes more significant than the gain in coherence from a few intersection tests (all primitives are in the CPU cache). Moreover, the efficiency of the FGR cache depends also on the parameter setting for the cache. As one can observe in Table 6.3, the optimal setting for the number of stratas depends on the number of FGRs. Optimizing the parameters of the cache is a difficult task since the speedup depends on several quantities: the scene model, the rendering setting (e.g. FGRs per pixel, image resolution), and the hardware (e.g. CPU cache size, processor type). The tests in Table 6.3 do not account for the size of the buffer in each

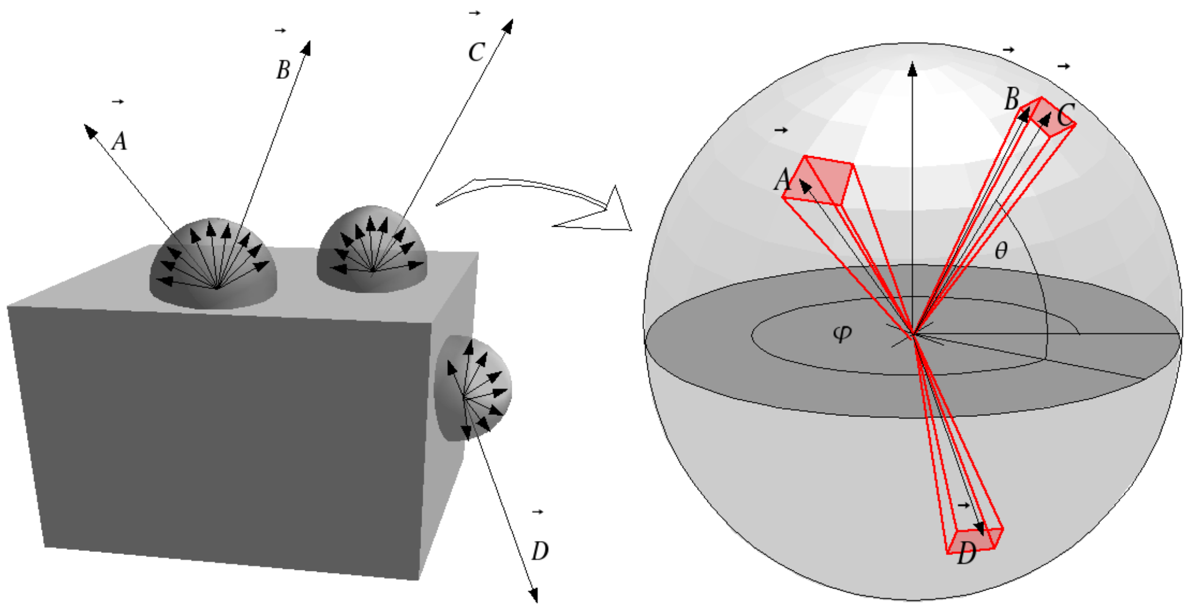


Figure 6.13: *Caching final gather rays (FGRs). The directions of the FGRs $\vec{A}, \vec{B}, \vec{C}$, $\vec{D}$ (left) are transformed to polar coordinates and mapped to stratas of a discrete spherical cache (right) independent of their origin. FGRs are accumulated in their corresponding stratas. When the array associated with the strata is full, all FGRs inside are traced at once. Note that the FGRs are only clustered in the directional domain independently of their origin (e.g. rays vecB and vecC).*

strata. We simply limit the buffer to 100 rays per strata. On average a smaller size (50 and 75 stratas were tested) performed worse.

The FGR cache has only one disadvantage that is: it does not run together with irradiance caching. For irradiance caching we need all final gather rays shot at once to compute the harmonic mean distance and if required the rotational and translational gradients. This is not easy to manage with the FGR cache where each FGR is processed irregularly, independent of time and location. Therefore, the FGR cache is automatically disabled when irradiance caching is enabled.

6.9.2 Aggregate Kd-tree Traversal

Another optimisation we have applied to the reverse photon mapping algorithm is the traversal of the reverse photon kd-tree for several photons queries at once. We find the nearby photons in the leaves of the photon kd-tree (which are already close in space) and aggregate them by clustering in one larger bounding sphere. During the clustering procedure the bounding sphere is extended until more than a maximum number of photons are clustered inside or until the sphere exceeds the maximum radius. The maximum radius is taken from the upper threshold for the gather radius (see source code in 6.15).

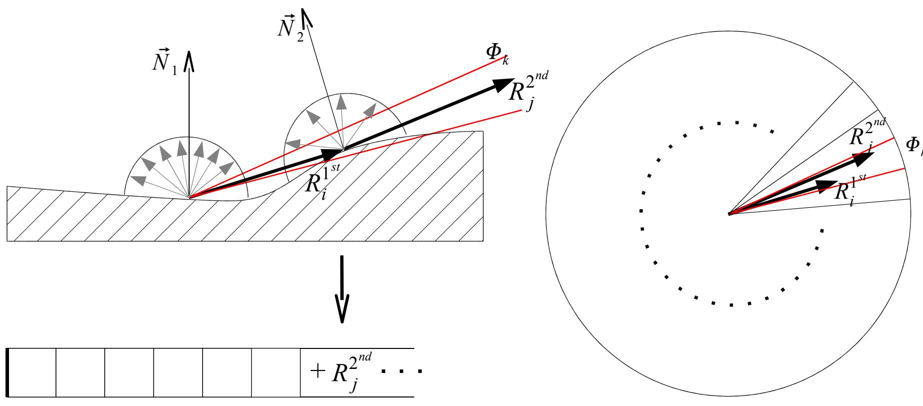


Figure 6.14: *Caching a secondary final gather rays $R_j^{2^{nd}}$ to the same strata Φ_k as its father ray $R_i^{1^{st}}$ when Φ_k is being flushed. Instead of adding $R_j^{2^{nd}}$ to Φ_k it is appended to a special queue only designed for this case since Φ_k is full and all its rays are processed.*

Scene	Res	*FGR*	T_{200}	T_{400}	T_{600}	T_{800}	T_{1000}	Naive	max. Speedup
Atrium	600^2	100	5,87	5,26	5,22	5,12	4,89	6,11	20,0%
Atrium	600^2	200	11,82	10,29	10,11	10,11	10,11	12,08	16,3%
Atrium	600^2	300	17,44	15,24	14,98	14,81	14,83	15,79	6,2%
Atrium	600^2	400	23,17	20,31	19,92	19,67	17,67	21,22	16,7%
Atrium	600^2	500	29,04	25,25	24,87	24,36	23,42	26,43	11,4%
Atrium	600^2	600	35,28	30,30	29,59	29,46	26,47	36,58	27,6%
Atrium	600^2	700	40,96	31,97	34,68	31,24	30,92	42,77	27,7%
Atrium	600^2	800	46,88	36,24	39,57	39,33	35,28	42,71	17,4%

Table 6.3: *Timings for final gather ray shooting including camera frame sampling and BRDF importance sampling for various number of final gather rays per hemisphere and 5 different resolutions ($\theta \times \phi$) of the final gather ray cache ranging from 200 to 1000 stratas discretizing the sphere. All stratas were set to cache 100 rays. The speedup compared to the naive single ray shooting is summarized in the last column. For the timings only a tile of size 50×50 with 7 samples per pixel was rendered. The importance sampled surfaces, visible within the rendered tile, involve glossy and diffuse BRDFs. Note, that the speedup is optimal for purely diffuse sampling over the hemisphere since it yields the highest variance.*

We traverse the reverse photon tree using this larger bounding sphere to find the first interior node whose assigned splitting plane cuts the bounding sphere. This is the starting node for the kd-tree traversal for all individual queries of the photons inside the bounding sphere. This aggregate search avoids repeated traversals in upper level nodes in the reverse photon kd-tree. The concept is visualized in Figure 6.16. The achieved speedup for the kd-tree search depends on the photon density and the depth of the reverse-photon kd-tree. We observed a speedup between 30 to 65 percent compared to traversing for individual photons. The average number of traversal steps in the kd-tree is reduced by more than 30% compared to individual queries starting from the root.

6.9.3 Density Control for the Photon Map

Using the adaptive radius is one method to reduce searching and processing time while decreasing the bias. Another adaptive technique that helps us to speed up the rendering, is density control

```
int rightIndex = 0;
while (rightIndex < nPhotons) {
    leftIndex = rightIndex;
    rightIndex++;
    const CPhoton& ph = photonMap->GetPhoton(leftIndex);
    radiusPh = ph.searchRadius;
    radiusBSphere = ph.searchRadius;
    searchSphere.Init(ph.pos, radiusPh);
    photonsInSphere = 1;
    maxRadiusReached = false;
    while ((rightIndex < nPhotons) && (!maxRadiusReached)) {
      const CPhoton& ph = photonMap->GetPhoton(rightIndex);
      radiusPh = ph.searchRadius;
      if (radiusPh > maxRadius || photonsInSphere >= _MAX_PHOTONS) {
        maxRadiusReached = true;
        break;
      }
      dist = Distance(searchSphere.pos, ph.pos);
      if (dist+radiusPh <= radiusBSphere) {
        rNew = radiusBSphere;
        t = 0;
      }
      // else spheres overlap or are disjoint
      // then calculate new radius and interpolation value t
      else {
        rNew = (dist + radiusPh + radiusBSphere) / 2.0f;
        t = (rNew - radiusBSphere) / dist;
      }
      if (rNew <= maxRadius) {
        rightIndex++;
        //compute new center of bounding sphere
        sphereCenter = ph.pos * t + searchSphere.pos * (1-t);
        searchSphere.Init( sphereCenter, rNew);
        photonsInSphere++;
        radiusBSphere = rNew;
      }
      else {
        maxRadiusReached = true;
      }
    }//end of second while
    int startNode = 0;
    bool foundSomething =
        eyePhotonMap->FirstIntersectedNode(startNode, searchSphere);
    if (!foundSomething) //we found empty voxel ->
      continue;
    // individual photon searches
    for (int i = leftIndex; i < rightIndex; i++) {
      const CPhoton& ph = photonMap->GetPhoton(i);
      searchSphere.Init(ph.pos, ph.searchRadius);
      eyePhotonMap->LocateEyePhotons(searchSphere, startNode);
      if (searchSphere.foundEyePhotons > 0)
        SplatPhotonEnergy(ph, searchSphere);
    }
  }
```

Figure 6.15: *The aggregate photon search in the reverse photon map.*

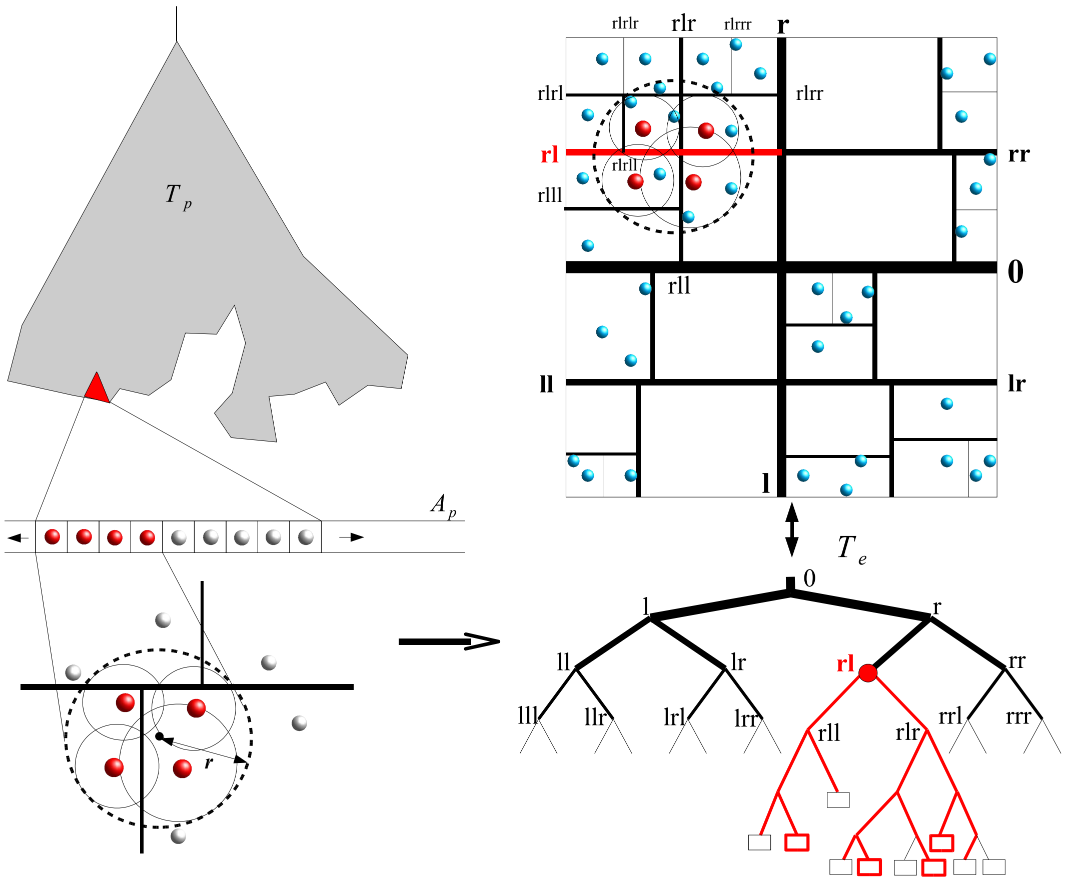


Figure 6.16: *An example for the aggregate search in 2D. Photons (red larger dots) from the sorted photon array (A_p) are clustered in a bounding circle and the reverse photon kd-tree (T_e) is searched for the first intersected splitting plane (thick red line). The corresponding node (rl) is the starting node for the individual photon queries (smaller circles).*

for the photon map. The concept is not new and has been studied in [77, 43]. However, rather than controlling the density on-the-fly during photon tracing, we propose a post-processing algorithm which is faster and easier to implement. It also introduces less bias since the discarded photons which we spread among their neighbors are randomly chosen with probability proportional to their density. Since the photons are sorted spatially, the linear photon array corresponds to a discrete cumulative distribution function (*CDF*), see Figure 6.17.

In the algorithm of Suykens and Willems [77] the discarded photons are determined by the sampling process which can induce a correlation that might lead to artifacts. On the other hand, our algorithm needs more initial memory because *all* photons are stored during the photon tracing phase regardless of the density. Suykens and Willems use the density control in combination with importance sampling. This helps to reduce the size of the photon map in *unimportant* regions (i.e. regions not contributing to the image). Importance sampling for the density control has not been applied to reverse photon mapping but is straightforward to implement since the ”importons” are inherently given by the reverse photons.

For reverse photon mapping density control is more decisive than for normal photon mapping since the time spent for density estimation strongly depends on the number of photons (see Table 6.1). This is because the photons are processed successively (if we neglect the aggre-

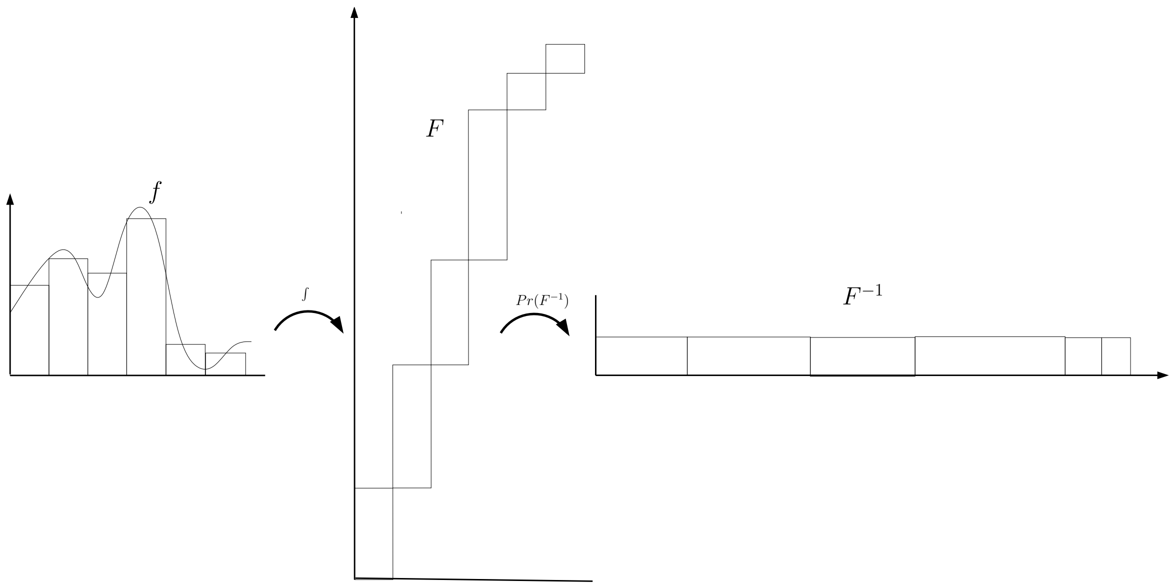


Figure 6.17: *Mapping a discrete probability density function (PDF) (left) to its cumulative distribution function (CDF) (middle) and its correspondence to the spatially sorted photon array (right). Each bar corresponds to the density within one voxel of a regular grid which is proportional to its number of photons.*

gate search, Section 6.9.2). For normal photon mapping, the number of stored photons is less significant with respect to the computation time since the time depends mainly on the number of final gather rays. Following, we roughly describe the algorithm used for density control in reverse photon mapping. The density control takes negligible time compared to the rest of the computation for reverse photon mapping.

We use an iterative randomized algorithm and we assume that the kd-tree over photons is constructed and the gather radius for each leaf in the photon tree is precomputed (see Section 6.5). A simplified version of the algorithm is listed in Figure 6.18. The algorithm 6.18 proceeds until a maximum number of iterations is exceeded (= number of photons) or less than a minimum number of photons N_{min} with higher density than the given maximum density D_{max} has been found in one pass. This additional constraint triggers an early break in cases when there is almost nothing to reduce any longer. Since the probability to pick a photon from a particular region is roughly proportional to its underlying density, it is likely to find high density photons within a limited number of iterations M. The probability $P(V_i)$ to pick a photon P_j from a voxel V_i with density above D_{max} within M iterations is:

$$P(V_i) = 1 - (1 - \sum_{i \in \Delta} p(V_i))^M \ , \ \Delta = \{i | Density(V_i) > D_{max}\},$$

where p is the probability density function (PDF) determined by the density of photons. For example, if one percent of all photons have greater density than D_{max} the probability P to find one in $M = 1000$ iterations is 0.99997. Without the early termination constraint the computation time of the density control can become considerably long since we would need to compute the density at each photon location and the gain in rendering speedup might be lost. To judge our method, we tested the brute-force approach: iterating over the entire photon array and computing the density at each photon location. Although the iteration is highly coherent, the performance of this approach is too slow for a large amount of photons. This is in particular a problem when none of the photon locations or only a small fraction of the photon locations

```
$N_p \leftarrow$ number of photons
MAX_PASSES $\leftarrow$ 100
$M \leftarrow N_p$ / MAX_PASSES
$N_{min} \leftarrow M$ / 1000
K_NN $\leftarrow$ 30
precompute $R_{search}$
$pass \leftarrow 0$
**repeat**
  $N_{distributed} \leftarrow 0$
  $pass$++
  **for** i $\leftarrow$ 1 to $M$ **do**
    pick random photon $P_j$ with $j \in [0..N_p]$
    **if** not $P_j$ marked as invalid **then**
      gather $K$ nearest valid photons $S_j$ around $P_j$ within $R_{search}$
      compute density $D_j$
      **if** $D_j > D_{max}$ **then**
        distribute energy of $P_j$ among neighbors using kernel $K_s$
        mark $P_j$ as invalid
        $N_{distributed}$++
      **end if**
    **end if**
  **end for**
**until** $pass$ = MAX_PASSES or $N_{distributed} < N_{min}$
remove invalid photons
rebuild kd-tree over valid photons
precompute the gather radii
**return** new number of photons
```

Figure 6.18: *The algorithm for density control for the photon map.*

has a "too high" density (e.g. caustics). We also experimented with stratified sampling over the photon array, which did not noticeably improve the algorithm.

The gather radius R_{search} is tuned to find on average a few photons less per query than the specified maximum number for K. As a result we avoid performing an expensive k-median sort on the gathered photons. The method works well in practice. We have chosen a constant number for K that is set to 30.

Suykens and Willems used a simple box filter to distribute the energy equally among the neighboring photons. We chose a filter function that gives most energy to photons that are close to the location of photon P_j. A common choice is the *biweight* kernel function (see Section 3.6.4.5). This kernel is a good approximation to the gaussian kernel and is cheaper to evaluate. We distribute the energy of photon P_j among the neighboring photons accounting for photons with similar normal and incoming direction on the same hemisphere. At the end, all gaps from invalidated photons are removed by traversing the photon array from left to right and filling the gaps with valid photons from the right. Finally, the kd-tree is constructed and the photon bandwidths (gather radii) are precomputed. The results for the photon density control are shown in Figure 6.19. A problem with density control is that it increases variance of the photon power. In combination with the precomputed gather radius for adaptive density estimation, the results may exhibit occlusion bias (light leakage) due to the automatically increased bandwidth in regions of reduced density. This problem cannot easily be eliminated since we loose information when decreasing the photon density. Simply keeping the precomputed bandwidth per photon assigned before density decimation results in visible noise. Nevertheless, applying density control with a sensible setting for the maximum allowed density, we save memory and rendering time with little loss in image quality.

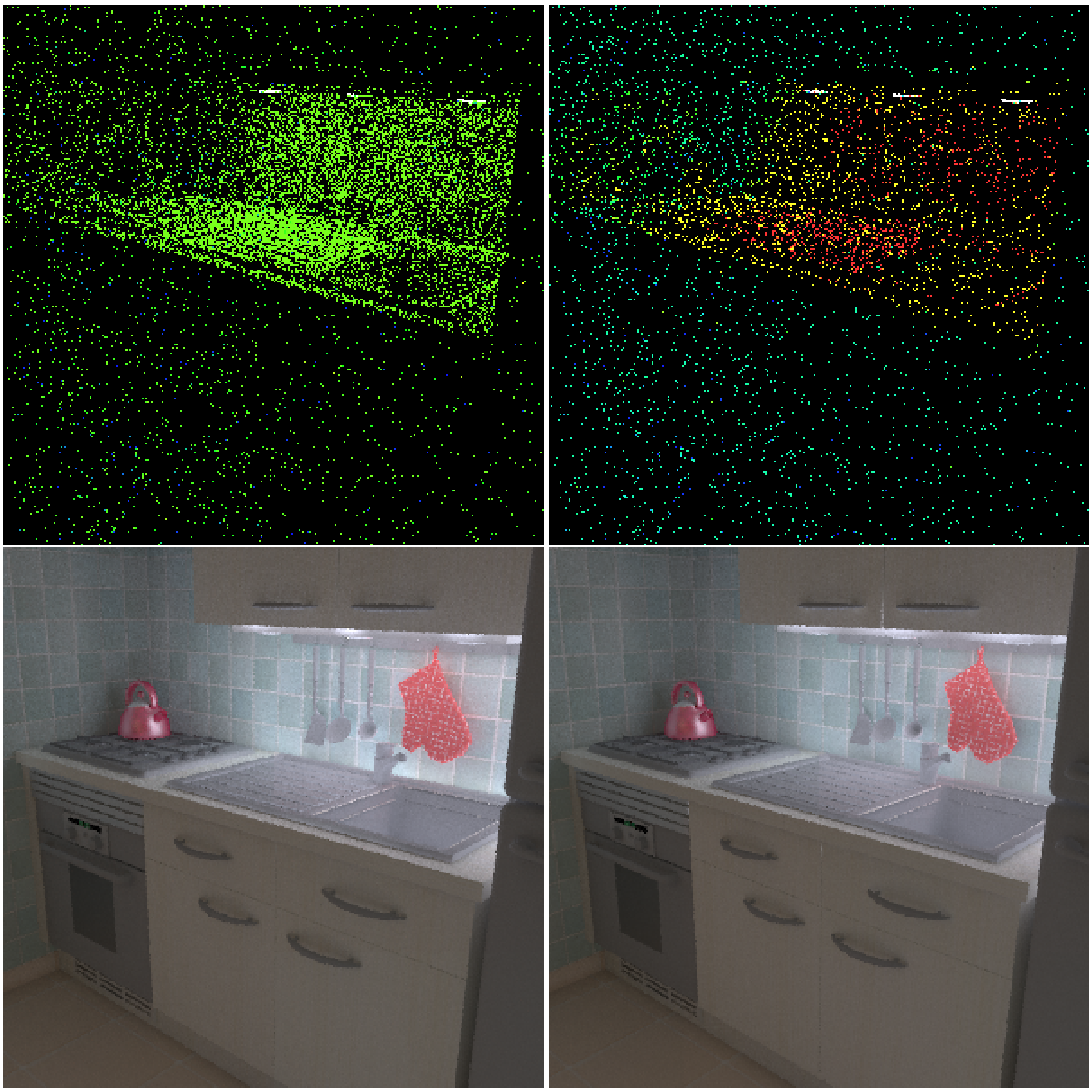


Figure 6.19: *False color images of the photon distribution in the kitchen scene using 700,000 photons without density control (top left) and approximately 370,000 photons with density control using a maximum density of* $10,000$ *photons/m*^{2} *(top right). The false color (from blue to red) corresponds to the energy of a photon. Note how the energy increases in high density regions after density control was applied. Below are the rendered images showing only indirect illumination using the photon map displayed above.*

6.9.4 Low-level Optimization

Besides all previously described algorithmic optimization techniques, there is one method that always works and should not be underestimated. This is code optimization. Due to the continual increase in processor speed, the bottleneck between CPU and main memory becomes more crucial. Hence, caching has become more and more important and gave rise to the cache-aware algorithms. In particular for algorithms working on large data sets such as reverse photon mapping, caching is very decisive. Therefore, we will briefly describe the modifications we applied to the data structures and the algorithm workflow.

6.9.4.1 The Write-Back Cache for Accumulating Radiance

The most significant optimization we applied to reverse photon mapping is the write-back cache for gathering radiance in each reverse photon. In the early stages of our algorithm we wrote the computed radiance for each reverse photon from a photon query directly to its corresponding screen pixel where it originated from. This method had the advantage of progressive previewing since the image was progressively refined and displayed during photon tracing. However, even if the reverse photons in a query are coherent in space and memory their corresponding pixels are not and splatting the photon's energy to the image becomes a random process. The image is considerably large and does not fit into the cache memory together with the photon query, which results in many cache misses. However, the gathered reverse photons from successive photon queries are highly coherent. This is the reason why we created a second array of the same size and same order as the reverse photon array, which we call the *cumulative radiance array* (A_w). The elements in A_w consist of only 3 floats and represents one radiance sample of a pixel in RGB. Thus, every element in A_w corresponds to exactly one reverse photon and has the same index as its reverse photon. As we sum the radiance in this separate array, the reverse photon array stays read-only for the entire search and density estimation phase (photon splatting). During the photon splatting, the radiance from all photons in the neighborhood of an reverse photon is accumulated in the reverse photon's corresponding "write-back cell" of A_w. This way, we establish coherency at the expense of using additional memory for the array A_w which is much more larger than the number of pixels in the screen (tile). Since A_w is around three times smaller than the reverse photon array, it is also too large to fit in the memory for the whole rendering phase. Therefore, we also need to delete and resize it for each tile (see Section 6.7) or write it to hard disk using the external cache described in Section 6.10. At the end, the array A_w is processed in sequential order. The accumulated radiance for each reverse photon is summed to the radiance along pixel samples which is possibly filtered by some convolution kernel before being added to the resulting image (see Figure 6.12).

Using the cumulative radiance array has another advantage. We can postpone the multiplication with the actual contribution of an reverse photon. Instead of computing the radiance contribution to a pixel sample for every individual photon, we accumulate the local radiance for each reverse photon and compute the total contribution (including the light transport via BRDF sampling to the pixel) at the end. This saves three multiplications in the most expensive inner loop for distributing the photon's energy that is usually iterated between 10^8 and 10^{11} times.

The achieved speedup using the cumulative radiance array is one order of magnitude (up to approx. 1000 percent). However, combined with the dual-tree approach storing and sorting all photons in space, the memory coherence is as high as possible for our algorithm. Therefore, the speedup compared with the naive method (splatting photon energy directly to screen) is two orders of magnitude.

We have also experimented with a write-back cache that accumulates radiance from photons in a small window until we access an reverse photon index outside the range of the cache window. The cache is then emptied by adding the accumulated radiance to the pixel samples and the window is moved in direction of the new access. This method is superior to the naive method (writing radiance directly to screen) but still a few times slower than having a fixed complete write-back array. It seemed that the overhead for moving the window and emptying and re-initializing the cache is too expensive.

6.9.4.2 Splitting the Reverse Photon Data Structures into Different Arrays

Another trick for improving cache coherence is the splitting of large data structures into temporal disjoint parts. In our case the reverse photon consists of 32 Bytes from which only 14 Bytes are accessed during processing phase (kd-tree traversal and photon splatting). The kd-tree traversal needs only the position (12 Bytes) and, in case of a kd-tree with photon-nodes (see Section 5.2), the splitting plane axis (1 Byte). The photon splatting uses the incoming direction (2 bytes) and the index inherently given by the reverse photon. Hence, only 15 Bytes (14 Bytes in case of the *kd-tree8*) are accessed during the processing phase. The remaining 16 Bytes for spectral weight in RGB or XYZ (12 bytes) and pixel sample index (4 bytes) can be removed and put to another array that is accessed only at the end in the completion phase. This results in two separate arrays each consisting of 16 Byte elements (due to alignment). In our algorithm, this technique yields between 2 and 5 percent speedup.

6.10 External Caching

We have already presented several data structures for the reverse photon map and normal photon map in Chapter 5. These were designed for efficient searching in the main memory. For reverse photon mapping (RPM) we have seen that the memory utilization is crucial and that the complete data does usually not fit into the main memory. We created a scheduler that subdivides the rendering into independent tasks. However, one drawback with this subdivision is that we cannot take advantage of the full capacities of reverse photon mapping since the search speedup increases with the amount of available memory. Hence, the more data (i.e. reverse photons) we can store in memory, the fewer tasks need to be created by the scheduler and therefore the fewer queries in the reverse photon map are performed.

Besides tiling and scheduling, we have another possibility to deal with the large memory requirements: external caching. As we mentioned before, the main memory (RAM) is usually limited by 1 to 4 GBytes. However, the external memory (i.e. hard disk) is about two orders of magnitude larger (nowadays between 40 GBytes and 400 GBytes) than the main memory. However, hard disks are mechanical instruments and the access time for a hard disk is in the order of milliseconds ($\approx 10 - 20ms$) whereas the access time for the RAM is in the order of nanoseconds ($\approx 50 - 100ns$). Furthermore, the data transfer rate (throughput) is two orders of magnitude slower than the throughput of RAM (modern RAM: ≈ 3000 MBytes/sec, modern hard-drives: $\approx 30 - 40$ MBytes/sec).

It would not be convenient to store all data to the hard disk and access it for each search query. We need to reduce the number of accesses to the hard disk to a minimum. Moreover, a hard disk read-request usually loads several successive KBytes (blocks) at once even if the requested data is only a few Bytes large. This means that the access pattern to the data must be highly coherent; in the ideal case, each Byte belonging to a data element (i.e. reverse photon) is read from disc at most once. Fortunately, our searches carried out in the reverse photon map are very coherent due to the spatial dual-tree approach. This allows us to create an efficient caching scheme that uses the hard disk. Only a small portion of the reverse photon array A_r and the write-back array A_w (see Section 6.9.4.1 for details) is cached in the main memory. We use a traditional software cache implemented via hashing, splitting the array into blocks of equal size. The size of the blocks is $2^{10} - 2^{15}$ Bytes and cache associativity is set to 8. A simple hashing function uses the index of a block on the hard disk to keep only a small fraction of the blocks in the main memory.

The array A_r is used in four steps: array creation and kd-tree construction, spatial searching, density estimation, and summing the results for FGRs to corresponding pixel samples (i.e. irradiance computation at visible hit points of primary rays). During creation of FGRs, we save the reverse photons to array A_r which is frequently flushed to disk when the internal cache is full.

The next step is the construction of the kd-tree. This time we benefit from our spatial kd-tree consisting of 8 Byte interior nodes on top of the reverse photon array. The kd-tree fits in the main memory even for the maximum allowed number of reverse photons (2^{30}) since the tree needs around 100 times less memory. This leads to at most ≈ 300 MBytes memory consumption for the whole kd-tree in the worst case. However, we need to modify the tree structure slightly in order to guarantee efficient searches. We do not want to evaluate the expensive hash function for each access to the reverse photon array and check whether the block containing the requested reverse photon is currently in memory or not. Therefore, we add a specific interior node (load-node) to the kd-tree that is only responsible for loading one (or two) requested block(s) from disk to memory. This node is inserted in the tree whenever the number of reverse photons in the current sub-tree is less than or equal to the size of a block. This ensures that each reverse photon in a leaf is located in memory when it is accessed during tree traversal. We only need to keep track of two pointers to two blocks and compute the offset with respect to the first and second block index (which is simply the reverse photon index modulo blocksize). Since a leaf is much smaller than a block, we nearly always access only one block during search in the leaf. However, there can be at most two blocks associated with the sub-tree of a load-node and we must occasionally check both of them if a leaf is located in both blocks. For each traversal from the root to a leaf of the tree there is exactly one load-node per path. The concept is shown in Figure 6.20.

During the reverse photon map (kd-tree) construction, we need to perform a binary sort of the reverse photons with respect to the splitting plane of the current kd-tree node. We do this by sequentially traversing the array A_r from left to right and from right to left until we find two entries that must be exchanged. The algorithm is very similar to the quick-sort algorithm, however the pivot element is given by the splitting plane position. This way, we only access each element once and one after another which creates a coherent access pattern to A_r and avoids irregular jumping between hash blocks. Since the size of a hash block is relatively large compared to the size of an reverse photon, the blocks are mostly loaded and written at the

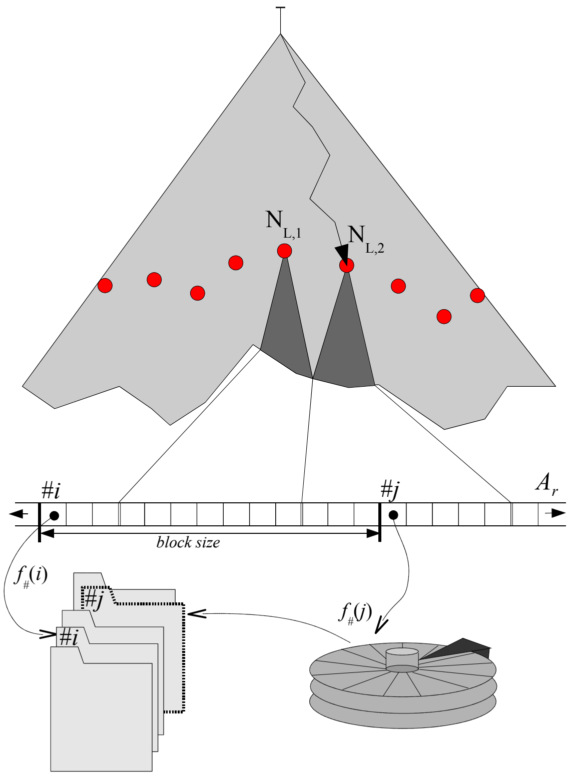


Figure 6.20: *Loading blocks $\#i$ and $\#j$ to main memory when accessing node $N_{L,2}$. Block $\#i$ was already read to memory by node $N_{L,1}$ and the hash function simply returns the address in memory. Block $\#j$ needs to be read from hard disk and mapped to a free address possibly replacing an old block. Note, that all entries in the sub-tree below $N_{L,1}$ and $N_{L,2}$ are located in main memory at addresses $f_\#(i)$ to $f_\#(j)+$ block size and there is no need to call the hash function $f_\#(x)$ at leaf-level.*

higher levels of the tree and sorting at the lower levels is mostly done in main memory where we access only a few blocks. Notice that for the external cache it is much more important that we store more than one reverse photon per leaf since this reduces significantly the number of sort operations at the lower levels of the tree.

During the search and density estimation phase, we need to keep only a small part of the array A_r and A_w in the main memory because successive photon queries into the reverse photon map are likely to access the same reverse photons and hence the same hash blocks. Thus, the hard disk accesses are reduced to a minimum. The observed performance slowdown during the rendering phase was between 2 and 10 percent only. The final step is the summation of radiance along FGRs stored in the write-back array A_w which we access in a sequential order using the indices and weights from A_r.

So far, our rendering algorithm performed surprisingly well but the bottle-neck moved to the kd-tree construction that took far too long for larger settings (up to 25 minutes for about 30 million reverse photons). In our current implementation of the external cache, we have to construct the whole tree from root to leaves above the reverse photons array which is stored entirely on disk. Let the number of reverse photons be N_r and the average number of reverse photons per leaf T_m. Then we have to read $\mathcal{O}(N_r \cdot \log_2(N_r/T_m))$ reverse photons from the hard disk and also to write them back. Let us assume we have stored $N_r = 2^{27} \approx 134$ million reverse photons, which is quite common even for normal settings, and $T_m = 16$. Then we need to access about $\log_2(2^{27}/16) \cdot 134 \times 10^6 \approx 3.1$ billion reverse photons. This yields a total hard disk throughput of $\approx 94.1 \times 10^9$ Bytes ($\approx$ 92 GBytes) to be read and written. Modern hard disks have an average throughput of about 30 - 40 MBytes per second. Hence, only the data transfer between main memory and hard disk takes about one hour and a half.

Another reason for the slow kd-tree construction might be due to the fact that we did not account for low-level system behavior such as buffering and page swapping. Hence, there might be space for optimization of disk accesses. In addition, more tests with different block sizes and perhaps also other operating systems should be made.

6.11 Results

In this section we present the results for reverse photon mapping and compare them with normal photon mapping. At the end, we show a full global illumination example for a complex scene rendered with reverse photon mapping.

We measured the rendering timings for 8 scenes of various complexity and lighting conditions. Rendered images of the scenes are shown in Figure 6.21. The images show only diffuse and glossy indirect illumination as carried out by NPM using the global photon map. Direct illumination and caustics are not included since they are computed in the same way for both methods. The computed images are virtually the same for normal and reverse photon mapping.

We have implemented NPM for comparison purposes using the same data structures as for RPM. Both methods use a recursive version for the construction and search in the spatial kd-tree. The timings for normal and reverse photon mapping are listed in Table 6.4. Whereas for RPM we used the adaptive bandwidth for density estimation, for NPM we applied the traditional kNN density estimation using the proposed spatial kd-tree over photons. In both cases we applied the density control resulting in 500,000 photons. The variation in the average number of the k-nearest neighbors ($\bar{K}$) comes from the adaptive bandwidth selection. The bandwidth for the adaptive density estimation was tweaked to find on average the same number of photons per reverse photon as in the k-nearest neighbors photon search. In practice, the total achieved speedup in favour of RPM varies between $30-230\%$ depending on the resolution and the number of final gather rays per pixel. Note that the acceleration methods for reverse photon mapping (irradiance caching and the ray shooting cache) were disabled in the tests for the sake of fair comparison.

We carried out unobtrusive profiling of the code execution. In normal photon mapping about 70% of the time is spent on the kNN search, which is decreases to about 50% for the kd-tree described in Section 5.2.1. About 15% is required for density estimation with a box kernel, 10% is taken by visibility computations (without direct illumination computation), and 3% for BRDF importance sampling. The remaining 2% are required by other operations.

In reverse photon mapping searching takes about 20-25% of the total time (without the aggregate search described in Section 6.9.2). About 37% is required for density estimation, 20% for visibility computations (without direct illumination computation), 8% for constructing the kd-trees, 6% for BRDF importance sampling, and the remaining 4% for other operations including task management and pixel filtering.

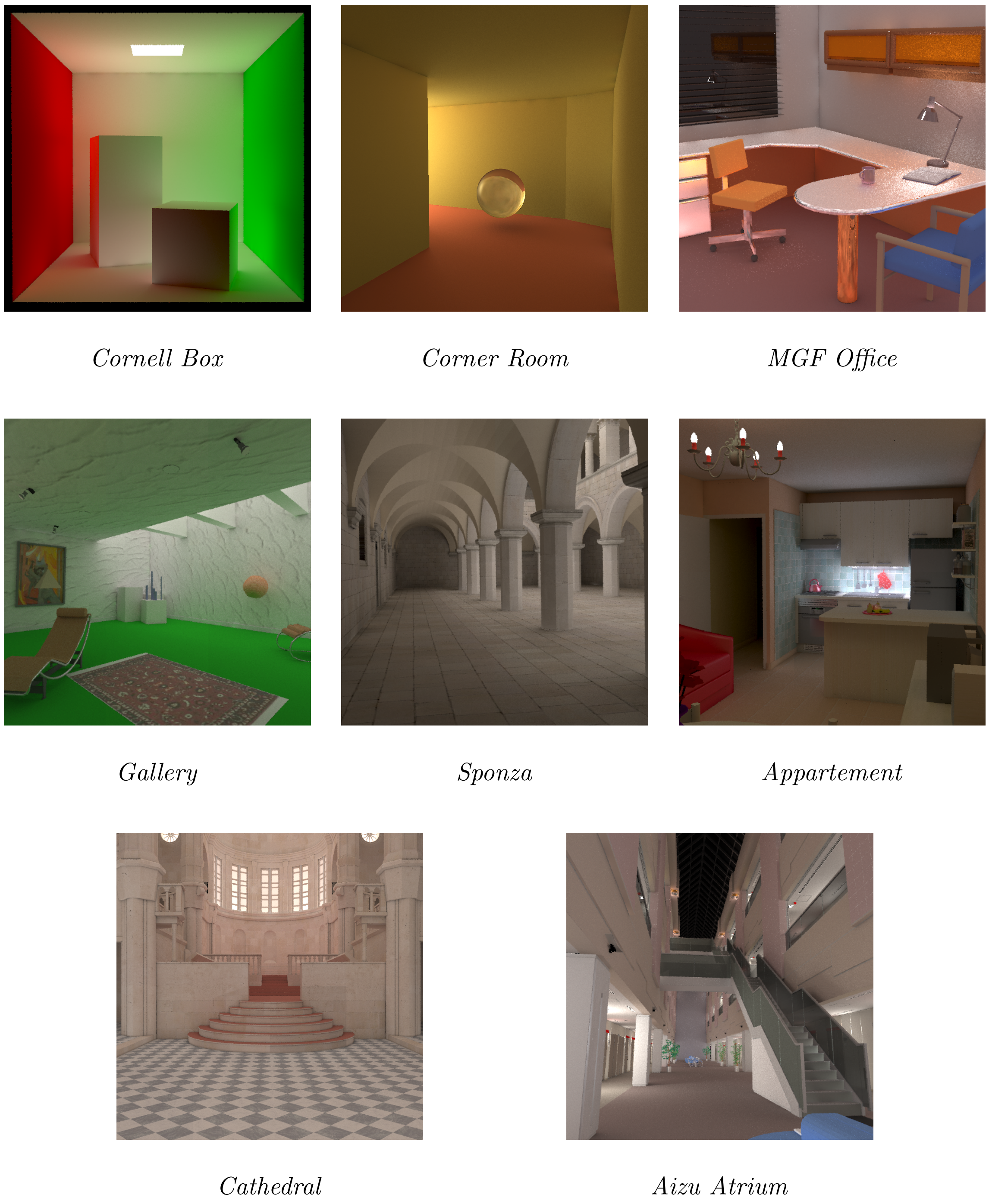


Figure 6.21: *Rendered images of the scenes used for the tests in Table 6.4. The images show only indirect diffuse and glossy illumination computed with reverse photon mapping. The images were rendered in a resolution of* 500×500 *pixels. In order to eliminate the noise in final gathering, 1500 to 3500 final gather rays per pixel were necessary depending on the scene.*

Scene	Res	$\bar{K}$	T^T_{NPM}	T^T_{RPM}	Speedup
Cornell Box	300×300	65	605	420	1,44
12 objs.	500×500	55	1540	710	2,16
Corner Room	300×300	30	1100	860	1,27
57 objs.	500×500	50	2200	1610	1,36
MGF Office	300×300	50	941	399	2,35
540 objs.	500×500	60	3600	1125	3,20
Gallery	300×300	50	1250	386	3,23
8607 objs.	500×500	50	2600	1248	2,08
Sponza	300×300	50	840	493	1,70
66650 objs.	500×500	50	2613	1280	2,04
Appartement	300×300	45	810	343	2,36
72270 objs.	500×500	45	2231	926	2,40
Sibenik	300×300	50	610	460	1,32
76643 objs.	500×500	50	1710	1250	1,36
Aizu Atrium	300×300	55	1793	541	3,31
948371 objs.	500×500	55	4712	1450	3,24

Table 6.4: *The timings for 8 scenes, using 600 final gather rays per pixel and resolution* 300×300 *and* 500×500. *From leftmost column to rightmost column: the scene name with the number of objects, the resolution of the image, the average number of found photons per final gather ray* $\bar{K}$, *the overall computation time for normal photon mapping* T^T_{NPM}, *and reverse photon mapping* T^T_{RPM}, *and the speedup achieved for reverse photon mapping.*

The time for searching using reverse photon mapping is efficiently accelerated by a factor of 4-7 compared to normal photon mapping. The time required for density estimation including kernel evaluation and feasibility test for occlusion is increased although the number of evaluations for pairs photon×reverse photon is almost the same. This is due to the increased rate of write-back operations from the CPU cache to the main memory (via array A_w).

If the aggregate search is carried out, the time required for searching is decreased by 30% to 65%. In total the speedup only for searching varies between 20 and 30 compared to the time for searching in normal photon mapping. The number of traversal steps in the reverse photon map was decreased by additional 35%.

The speedup for ray tracing of FGRs using the proposed ray shooting cache varied between 5-30% compared with standard final gathering.

We computed an image from the *sponza* scene in a resolution of 2 Mpixels with 600 FGRs per pixel (1.2×10^9 final gather rays for the whole image). Such setting is commonly used in the production rendering [78]. Although the timing is in total 160 minutes (70 minutes for similar scene complexity in [78]), we compute more bounces of indirect illumination than in [78]. Our timings are comparable to the rendering with the irradiance atlas [10] ($\approx$ 200 minutes for 73×10^6 final gather rays for a complex scene).

For the measurements we have used a standard PC with a single CPU 3.0 GHz Intel P4 with 512 KBytes of L2 cache and 2 GBytes of memory. The program was implemented in C++ and compiled with GNU g++ version 3.3 with -O3 optimization.

Finally, we present a high quality global illumination image rendered from the icido scene. Figure 6.22 shows the individual light contributions from direct illumination (top), diffuse and glossy indirect illumination (middle), and specular indirect illumination (top and middle) combined in the final image (bottom). The image was rendered in a resolution of 640×480 pixels with $6,000$ FGRs per pixel ($5 \times 1,200$ samples) and $700,000$ photons in the global map and

Rendering Phases	Time [h:m:s]	[%]
Direct illumination:	3:54:51	53.33
Shooting final gather rays:	1:29:14	20.30
Reverse photon map construction:	0:19:39	4.49
Photon shooting:	0:00:30	0.11
Photon map construction and density control:	0:01:45	0.40
Density estimation (indirect illumination):	1:33:20	21.13
Density estimation (caustics):	0:00:59	0.22
Filtering and pixel reconstruction:	0:00:02	0.02
Total time:	7:20:20	100.00

Table 6.5: *Results for the icido scene shown in Figure 6.22. Note that the final gathering and direct illumination overwhelms the final result. The photon shooting phase with photon tree construction, density control, and gather radius precomputation is only done for the first tile and reused for all 125 successive tiles.*

$200,000$ photons in the caustic map. The total number of used final gather rays was 1.76×10^9 (stored hit point in the reverse photon map). This setting was necessary to suppress the noise in the final image due to the high-frequency illumination. For the adaptive bandwidth we used the precomputed gather radius that aims to find on average 50 nearest photons per photon location. For the caustic rendering we used our spatial kd-tree (Section 5.2.1) and k-nearest neighbors density estimation with k set to 200. The timings of the individual rendering phases are shown in Table 6.5. Note that the majority of the time was taken by the computation of the direct illumination (4 hours) since we did not apply any importance sampling technique. The indirect illumination computation time took less than 3.5 hours including final gathering, photon tracing, global and caustic photon map construction and bandwidth precomputation, search in the reverse photon map, and density estimation. This is acceptable for such a setting since we have not applied any approximation technique such as irradiance caching or irradiance precomputation. We also rendered the same setting of the icido scene with irradiance caching. However, non-diffuse BRDFs and high-frequency illumination make the scene not suitable for standard irradiance caching. The results exposed visible artifacts in places where the illumination changes rapidly even for a low error setting.

6.12 Future Work

Our proposed reverse photon mapping algorithm is a powerful concept and we claim that there is more to exploit than in the normal photon mapping algorithm because of the additional information about the view dependent eye pass. Therefore, the algorithm suits perfectly to most importance sampling techniques for direct illumination [90, 72, 6] and indirect illumination [43, 62, 77]. It is also straightforward to extend the proposed irradiance caching scheme to account for irradiance gradients [94] and radiance interpolation on glossy surface [75] via FGR reprojection.

One of the biggest advantages over normal photon mapping is that we can have much faster and more flexible bandwidth selection because the bandwidth can be precomputed for all photons independent of the reverse photons. For instance, we could detect large illumination gradients by applying differential checking [34] of the density during bandwidth precomputation per photon. For normal photon mapping we are obliged to choose a gather radius for every final gather

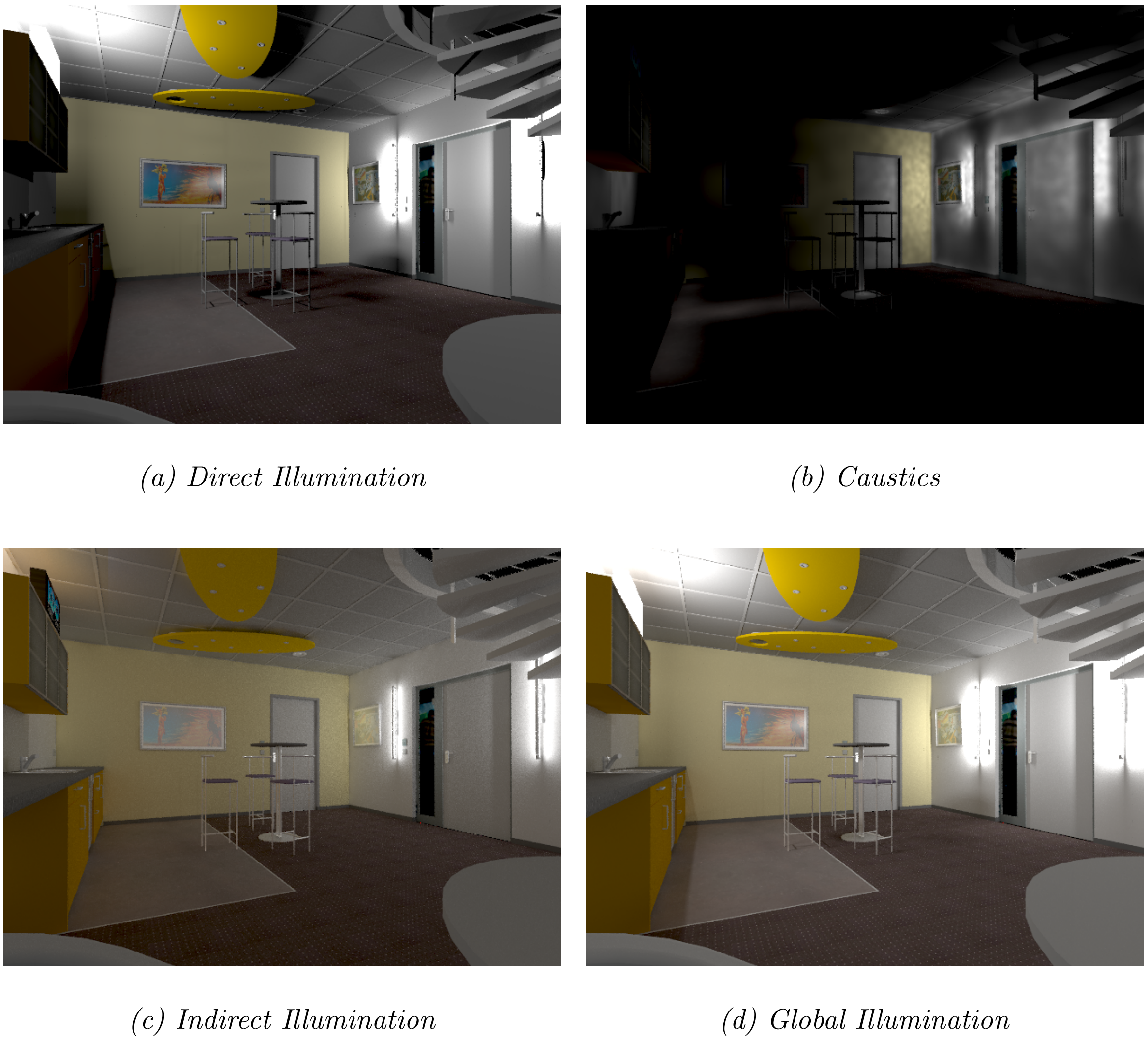


Figure 6.22: *Results for the icido scene rendered with reverse photon mapping. Image (d) shows the combination of direct illumination (a), caustics (b) and indirect diffuse/glossy illumination (c). Since the caustic contribution is too low, the caustic image was linearly scaled by a factor of 4 for visualization purposes. The indirect diffuse illumination was rendered with more than* 1.76 *billion stored reverse photons in 126 tiles with* 6,000 *final gather rays per pixel.*

ray which is far too expensive due to the incoherence and number of final gather rays. As a compromise, one usually sticks to the "cheap" kNN technique with a fixed k. This way the search domain for the density estimate is limited to a symmetrical sphere or box search. However, with reverse photon mapping which is based on statistical KDE the density estimation area for a final gather ray can have any shape since each photon uses an individual bandwidth to "spread" its energy to the final gather rays. It is also possible to extend the single bandwidth to a full bandwidth matrix giving control over the x-y-direction and the orientation of the energy spread for each photon.

Further, since reverse photon mapping is superior to normal photon mapping algorithms, we claim that it is extendable with any other technique known from photon mapping. Finally, it could be extended for rendering animations in the spirit of recent techniques that explore temporal coherence [80], which could bring additional speedup by one order of magnitude.

Chapter 7

Ray Maps

So far we have described how the density of photon hits is related to irradiance in a scene and how they can be stored in an appropriate way for searching. We have also presented several data structures for efficient storage of the photon hit points with the scene and introduced a novel "two-pass" algorithm in the previous chapter for more efficient computation of indirect illumination using the *popular* photon map.

We have shown the problems arising from photon mapping that impose a Monte Carlo sampling step called *final gathering* before the photon density estimation in order to diminish the visible error or replace it by less observable noise. As we have seen in the previous chapter, this technique is algorithmically expensive no matter what we are optimizing. Therefore, we will now focus on a new fundamental method for computing the illumination rather than optimizing and approximating the principles of the original methodology.

The method discussed in this chapter deals with organizing rays in space and has been developed in cooperation with Vlastimil Havran and Jiří Bittner. It was published to Eurographics Symposium on Rendering in 2005 [28].

7.1 Overview

In this chapter we will present a novel data structure for representing light transport in a scene called *ray map*. The ray map extends the concept of photon maps: it stores not only photon impacts but the whole photon paths represented by a sequence of rays. Ray maps are used for density estimation similarly to the photon maps, however, ray maps represent 4D information while photon maps store only 3D information about the light transport. This avoids boundary bias and improves the direct visualization without final gathering. We present a case study for the density estimation over photon maps and ray maps on a set of simple scenes. We propose a particular representation of ray maps using a lazily constructed spatial subdivision based on kd-trees. Additionally, we present several optimization techniques bringing the ray map query performance close to the performance of the photon map.

7.2 Previous Work

There are numerous data structures for storing illumination. A prominent example is the photon map [33] that stores light flux at the object surfaces, more specifically each point in the photon map represents a photon carrying certain energy. Photon mapping is described in Chapter 4. Other representations for indirect illumination include irradiance gradients [94], light maps [97], the line-space hierarchy [16], light vectors [98], or the irradiance volume [22].

The work presented in this thesis is closely related to the density estimation for photon maps. The density estimation was introduced to computer graphics by Heckbert [30] who first recognized that the computation of illumination from particle hits corresponds to density estimation [74, 89]. The result of the density estimation always induces a systematic error, referred to as bias. Every density estimation technique trades off between noise and bias. As discussed in Section 4.3.2 there are four basic types of bias (Figure 4.1) in the context of photon maps [76, 69]. For deeper discussion on bias for photon maps and an automatic bandwidth selection technique see Chapter 3 and Chapter 4. The error of lighting reconstruction of photon hits was studied by Myszkowski [58]. Several techniques have been proposed to improve the basic photon mapping algorithm, more or less intended to decrease the boundary bias. Hey and Purgathofer have dealt with the boundary bias using the average computed from several oriented photon maps [31]. Lavignotte and Paulin extend the object boundaries for storing of photons in polygonal scenes [49]. Several optimization techniques for photon maps such as the density control of photons were presented by Suykens and Willems [77] and Peter and Pietrek [62]. An extension of photon maps to account for the temporal domain was introduced by Cammarano and Jensen [7].

Below we discuss in more detail a conceptually different method first introduced by Lastra et al. [47] to remove boundary bias where the photon impacts are replaced by photons paths. The algorithm by Lastra et al. computes density estimation from photon paths intersecting a disc on the tangent plane. The estimation by using photon paths intersecting a disc instead of photon impacts efficiently reduces boundary bias inherent to photon maps. In order to search the nearest photon paths efficiently they build a ray cache that they call *dynamic list of spheres*, which is a hierarchy of spheres with decreasing radius. Each sphere is associated with a list of intersecting rays. At the highest level in the hierarchy the sphere contains all rays and at the lowest level the sphere radius is only slightly larger than the disc radius and the search for the candidate rays intersecting the disc is reduced to the smallest "bounding" sphere enclosing the disc. The concepts strongly relies on the assumption that successive density estimation queries are highly coherent in space such that each consecutive disc query can reuse mostly the same list of spheres from previous queries. Otherwise when a new disc query is outside the current sphere list, the list has to be updated from bottom to top. In order to obtain efficient results their data structure requires parameter settings for the ratio of consecutive sphere radii in the hierarchy and minimum sphere radius. These parameters strongly depend on the scene and render setting. Hence, their algorithm is penalized by two orders of magnitude increase of computational time compared to photon maps.

We will build on this concept but utilize more sophisticated data structures for ray proximity queries than a dynamic ray cache stored in a list of spheres. In order to solve high-dimensional complex problems efficiently, it is always advantageous to apply "divide and conquer" strategies. Spatial subdivision for efficient searching can be understood as such a kind of strategy. There is a rich literature on spatial data structures outside the computer graphics community [18]. A popular data structure for spatial subdivision and point or object indexing is the kd-tree [67]. The dynamization of kd-trees was introduced by van Kreveld and Overmars [82] decomposing the tree to subtrees according to individual dimensions. Proximity data structures for collision detection that are inherently dynamic were surveyed by Lin and Manocha [51] and Jimenez et al. [36]. Ar et al. [2] propose to construct BSP trees lazily for collision detection. We will propose an efficient data structure represented by a lazily constructed kd-tree for spatial subdivision of photon rays.

7.3 Representation of Light Paths

The ray map stores information about light paths traced during a global illumination simulation. The ray map itself is independent of the actual global illumination algorithm, that is the choice of the paths as well as the number of paths and the information associated with each ray on the path. The ray map represents exactly the same information as the photon map. However, for photon mapping the surface hits are computed in advance and the paths are discarded afterwards. Hence, the major difference is that the ray map also organizes the photon paths and not only the photon impacts on surfaces. Nevertheless, as for photon maps the density estimation is still a 2D problem but for the ray maps the photon hits are computed online during density estimation for a planar proximity domain. This domain is not on an arbitrary surface whose area is difficult to estimate but a ”perfect” shape (e.g. disc, sphere surface). Therefore, density estimation becomes entirely independent of the geometry and additional geometry-based bias sources are almost completely removed.

The problem of proximity search in the photon maps has dimensionality three, since the search works basically over 3D point data. The problem of proximity search in ray maps has dimensionality four, since the line data in the search is 4D. This results in an increased computational complexity and/or memory requirements for proximity searches. In Section 7.6 we address the problem of the increased computational complexity by enforcing the queries to be coherent so that the running time and memory requirements of the ray map are kept low.

7.3.1 Ray Proximity Queries

By organizing the paths the ray map can be used to answer queries that cannot be efficiently computed using the photon map. We can determine photons passing in proximity of an arbitrary point in space or all photons passing near the boundary of an object independently of the distance of the actual photon impact.

We distinguish between two different classes of queries: *intersection queries* and *nearest neighbor queries*. The intersection queries determine all rays intersecting a given spatial domain of fixed size. The nearest neighbor queries find k nearest rays using a particular distance metric. The queries can use different spatial domains and distance metrics:

I. **Intersection**. Domain:

- Disc
- Hemisphere
- Sphere
- Axis aligned bounding box

II. **K-Nearest Neighbors**. Metric:

- Distance to the point of intersection of the ray with a tangent plane

– Distance to the supporting line of the ray

In practice we should be able to use combinations of queries from the two classes. That is we should determine k nearest neighbors under a condition that they intersect the given spatial domain. However, not all the combinations of searches are justified in the context of density estimation.

In Figure 7.1 we show two important query methods that we use for density estimation. Figure 7.1(a) shows the standard query that searches the k nearest ray intersections (here $k = 4$) with the tangent plane. Ray 5 is farther away in the tangent plane than ray 4 even if its photon hit with the surface is closer. In the right (Figure b), our new *hemisphere-disc intersection* method is used: a ray is only included if

1. it intersects the hemisphere expanding around the density estimation point with radius R
2. and its prolongation also intersects the disc in the tangent plane.

Figure 7.1.b shows 7 different cases for ray intersections. The 1. ray is the standard case as shown in (Figure a). The 2. ray is included in the density estimation even if its surface hit is outside the hemisphere. The 3. ray ends in front of the tangent plane but intersects the hemisphere and its prolongation intersects the disc. The 4. ray also intersects the hemisphere but its prolongation not. The 5. ray is inside the hemisphere and is valid since it intersects the disc. The 6. ray does not even intersect the hemisphere and the 7. ray is actually occluded but since we do not detect this sort of occlusion bias, it is the same as case 1. As for normal photon mapping, all rays intersecting the tangent plane from below (i.e. have a positive dot product with the plane normal) are excluded from the density estimation.

The *hemisphere-disc* metric is a heuristic and cannot always guarantee a better density estimation than with the photon map. However, it usually leads to better results than with the pure disc query and the unnatural darkening at convex corners is reduced. We justify using this metric and intersection domain in the ray map in the following way. For searching the rays we use a hemispherical domain. However, we count only the rays that after prolongation intersect a disc. For the weight in the kernel we can use both: the distance in the tangent plane or the distance to the supporting line. First, this method is consistent with the rendering equation formulated for photon maps over the disc. Second, the approach removes boundary bias completely. Third, the method reduces topological bias, since only the rays intersecting a disc are taken into account, which is the assumption in the density estimation formula. The proposed search method is a key for implementing bias reduction techniques for density estimation. We support our selection of the method by experimental results on simple scenes in the next section. The images in Figure 7.2 show the density estimation footprints (i.e. ray/point candidates included in the density estimation) for three different intersection metrics: the standard photon map query (left), the ray disc query (middle), and the hemisphere-disc query (right).

7.4 Density Estimation

The ray map allows us to design a novel density estimation technique which makes use of a combination of metrics II.(a), II.(b), and II.(c). We use a k-nearest neighbors search which takes a maximum of the distances given by II.(a) and II.(b), i.e. the distance to the point on the tangent plane and the distance to the ray segment. Accordingly, either the distance to the line of

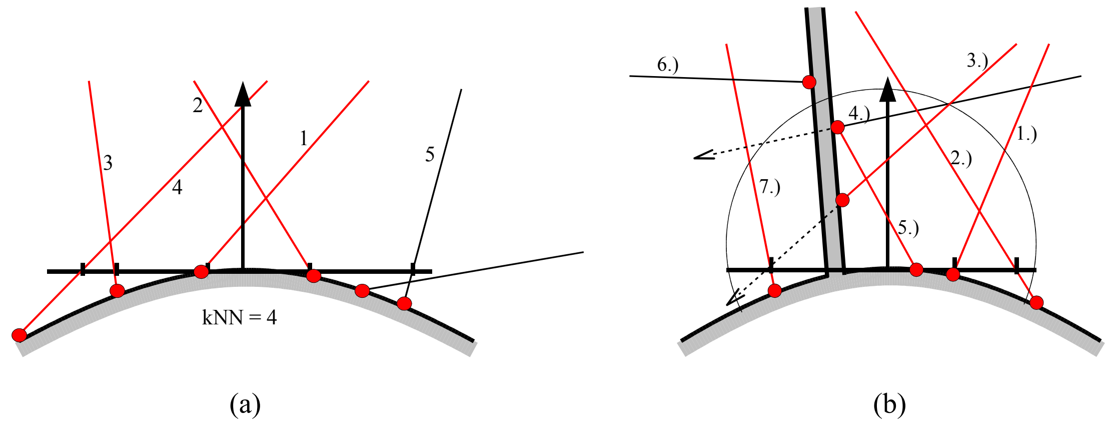


Figure 7.1: *The Figures (a) and (b) show different intersection metrics: Four nearest neighbors according to the euclidean distance in 2D of the intersection of a ray with the tangent plane (a). Note that due to the planar intersection domain, different photons can be included than for the 3D euclidean distance used for the photon map. The right figure (b) shows a complex query domain where the* hemisphere-disc intersection *metric takes effect (i.e. a ray is included if it first intersects the hemisphere and its prolongation (dashed line) intersects also the disc in the tangent plane). Figure (b) shows seven different cases for ray intersections. The red circles represent the photon hits with the surface. The red-colored rays are valid and the black ones are excluded.*

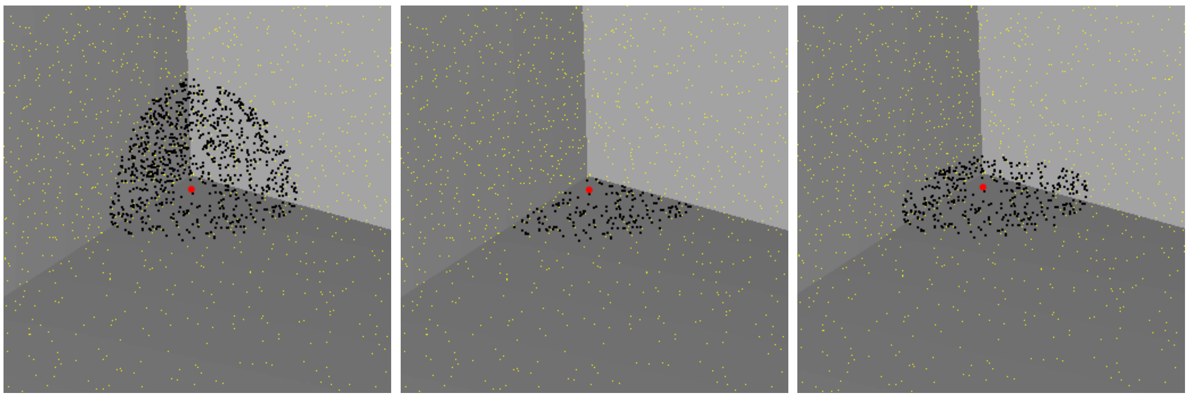


Figure 7.2: *The density estimation footprints inside a simple box for the "normal" photon map (left), for the ray disc intersection (middle), and for the* hemisphere-disc *method (right). Yellow points represent the photon hits (i.e. ray end points) and the black dots show the rays (photons) used for the density estimation at the red point. The sample of photons is drawn from a parallel light distribution starting from the opposite corner. (Notice that the red point is closer to the camera than the corner.)*

the ray or the distance in the tangent plane is used as a weight for the density estimation kernel. In this section we present the case study for density estimation we have made for ray maps and photon maps on a set of simple scenes and ray distributions. The simple scenes depicted in Figure 7.3 and Figure 7.4 were constructed in order to model the basic geometric features occurring in normal scenes.

7.4.1 Experimental Evaluation

Using photon maps the irradiance $E(x)$ at a point x is computed as:

$$E(x) \approx \sum_{i=1}^{K} \frac{\Delta\Phi(x, dA, \omega_i, d\omega_i)}{\Delta A} \tag{7.1}$$

$$\Delta A = \pi \cdot R^2 \tag{7.2}$$

This is equivalent to density estimation in case of diffuse (Lambertian) surfaces since the BRDF is constant and moved out of the sum. We have used the following four methods for density estimation:

- the reference irradiance computed either analytically or by a reference algorithm from two magnitudes higher number of photons following the same distribution,
- the irradiance from density estimation using the photon map,
- the irradiance computed from the ray map using the disc only,
- the irradiance from density estimation with the ray map using the hemisphere-disc metric,
- and for comparison with the hemisphere-disc metric, the irradiance computed from the ray map using the convex hull of the ray intersections with the disc to estimate the correct area in corners,
- partially, for comparison only, specific estimators were applied to the photon map: the adaptive KDE with precomputed bandwidth from the kNN for each photon hit, and the convex hull of nearest photon hits (the convex hull estimate for photons as well as for ray intersections is further discussed in 7.5 and 7.4.2).

In order to model basic geometric features occurring in rendered scenes, we implemented a program specifically designed for the analysis of 2D density estimation in a 3D model. Thereby, the user is able to visualize the density estimate along an arbitrary selected path (or 2D grid) on a surface using the photon map or the ray map. A graphical user interface allows for convenient construction and visualization of simple abstract scenes via OpenGL and their combination with various ray-photon distributions. This allows to simulate different illumination such as diffuse indirect light, caustics, or direct illumination from various light sources (e.g. point light, area light, parallel light source).

Figure 7.3 and Figure 7.4 show five selected test scenes from the analysis together with the associated ray-photon distribution (white lines). These scenes exhibit boundary and/or topological bias, which enables us to compare the properties of different density estimation methods and draw conclusions about the utility of the ray map. Just a small portion of the rays is shown for visualization purposes. The right column shows the 2D density estimates at the red dots along the depicted line (blue-red) on the left for various estimators using the photon map or the ray map. For the density estimation we used the Epanechnikov kernel (Section 3.6.4.4). For the tests two different ray-photon distributions were applied: the parallel light distribution with uniform density across a planar surface, and the diffuse light source distribution where the outgoing direction of a photon is proportional to the cosine of its azimuth angle with the normal of the light surface. The diffuse light source distribution approximates diffuse indirect light of scenes consisting mainly of Lambertian surfaces. For the diffuse light simulation photons were shot from a sphere surrounding the test scene and for the parallel light the photon origin was randomly chosen on a quad covering the whole test scene.

For the simple cases among the test scenes, an analytic estimate (black dotted curve) of the density along the depicted line was computed. For these cases we either assumed that no shadowing of photon rays can appear (i.e. the solid angle covered by the "light source" is not (partially) occluded at the density estimation point) or the light comes only from one direction (i.e. parallel ray distribution). For "artificial" directional light occlusion can be detected by one shadow ray since all other directions in the hemisphere do not contribute. For the parallel ray distribution the density (irradiance) at a certain point is simply the density of rays (surface area of emitter divided by number of samples) multiplied by the cosine of the ray direction with the normal at the surface point.

For more complicated test scenes exhibiting self-occlusion such as the wave surface in Figure 7.4 (middle and bottom) it is difficult to estimate the correct solid angle for arbitrary ray distributions (e.g. cosine ray distribution). Therefore, the reference estimate (black dotted curve) was computed from two orders of magnitude higher number of samples ($\approx 4 \times 10^6$ photons) following the same distribution. In addition, the surface boundaries were extended to overcome the boundary bias problem and adaptive bandwidth selection was used based on the ray/photon density. For the actual test 100,000 rays were drawn from the simulated light source.

7.4.2 Convex Hull of Ray Disc Intersections

Another possibility for improving the density estimation by reducing the topological bias is to compute the density estimation area from the convex hull of ray intersections with the disc. Unlike the *hemisphere-disc* intersection metric, the convex hull is not a simple heuristic and can lead to better density estimation in corners. Since we have a perfectly planar intersection domain (disc), the convex hull area estimate does not suffer from the usual problems that occur for photon mapping where we have to deal with a 3D domain. Moreover, unlike the photon map query domain, the intersections of rays with the disc mostly form a convex polygon (no geometric boundaries) and do not lead to area overestimation at concave surfaces. However, the problem of overestimation due to a polygonal approximation of the area (Figure 7.7 (a) and (b)) still appears for ray–disc intersections as well. Therefore, noise appears in low-density regions or areas where the convex hull becomes very coarse. Therefore, our *hemisphere-disc* method might be preferable since it is more robust. Figure 7.5 shows an example of the convex hull density estimation for a simple scene on the left. The density is uniform across each plane and the estimate was computed along the depicted line. On the right is a chart showing the density estimates for three different kernel methods. The yellow line is computed from the photon map via "normal" k-nearest neighbors and shows the typical bump near the corner. The dashed purple line is the estimate using the convex hull of the photon hits, which overestimates near the corner. And the dotted blue line shows the convex hull estimate for the ray map using the

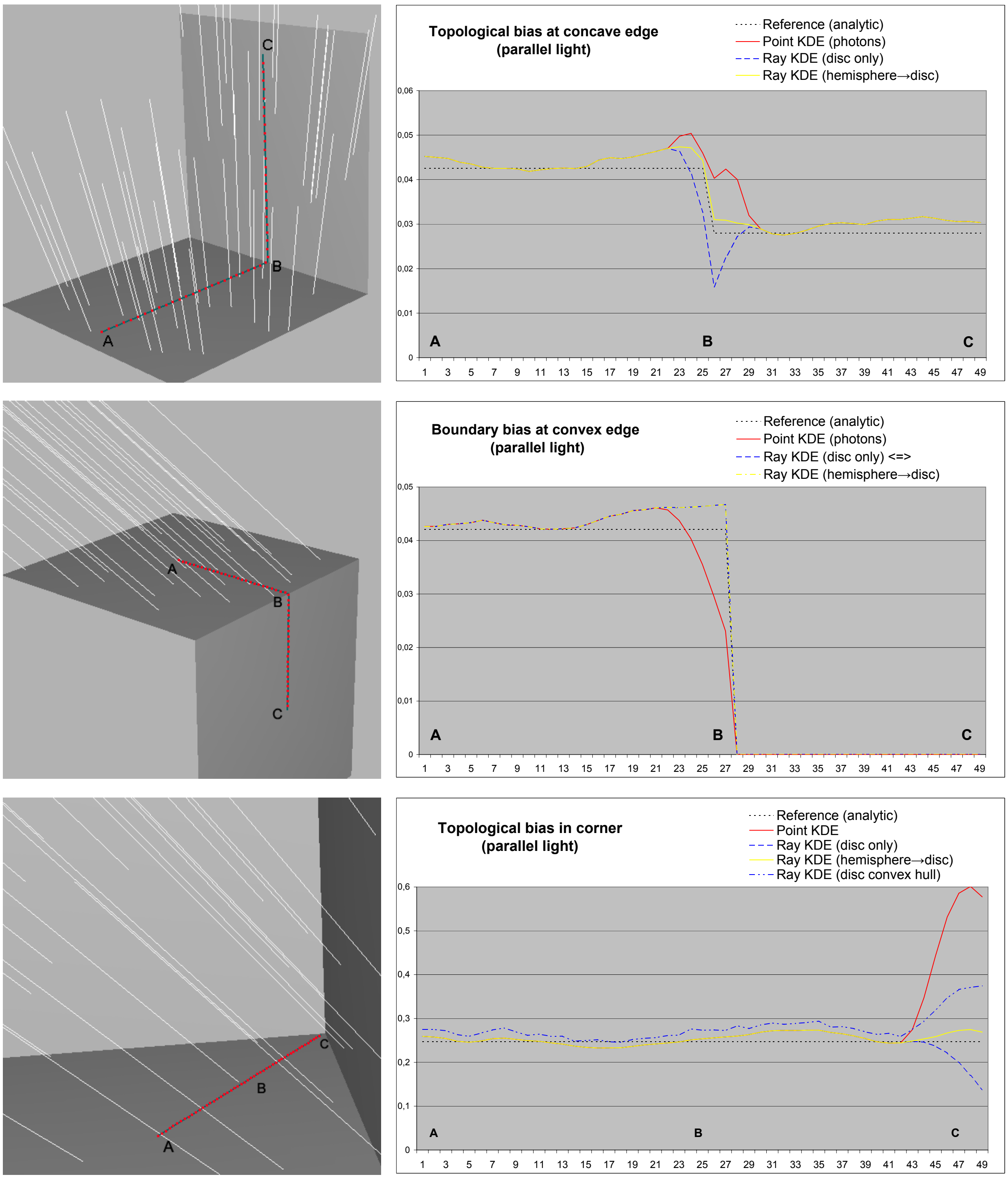
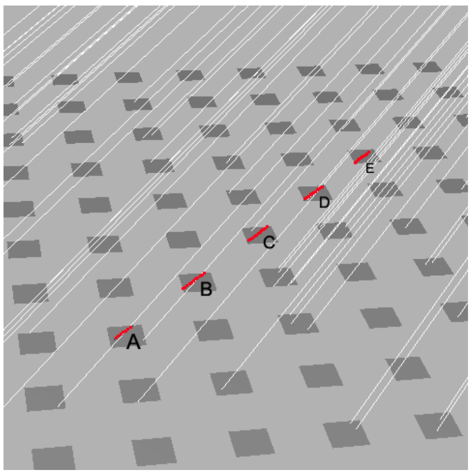
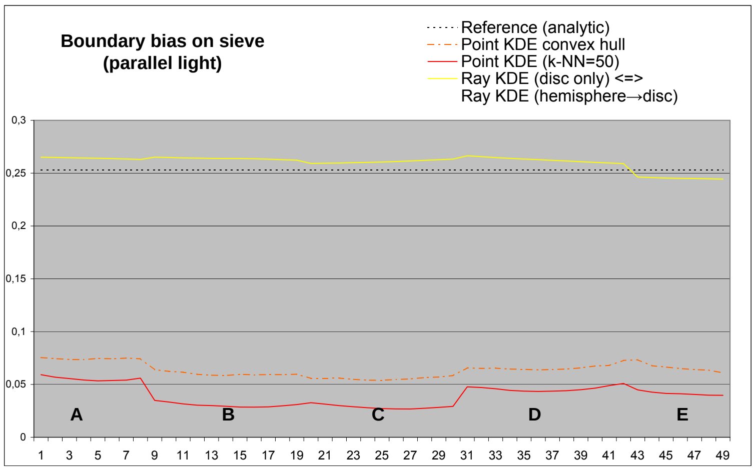
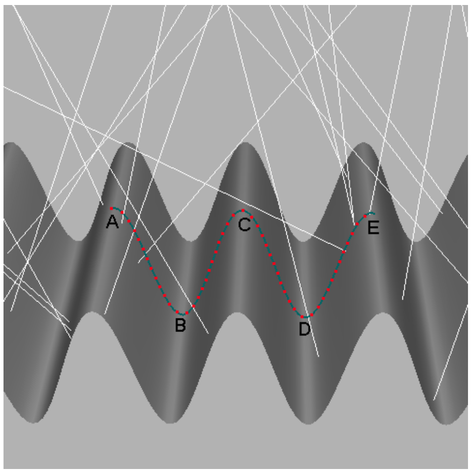
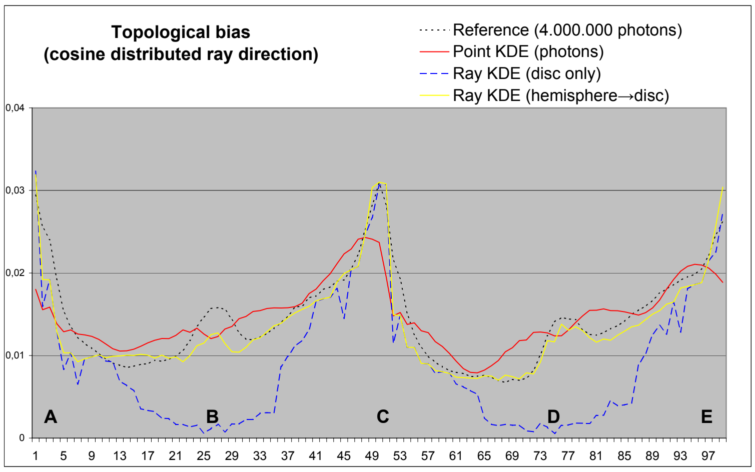


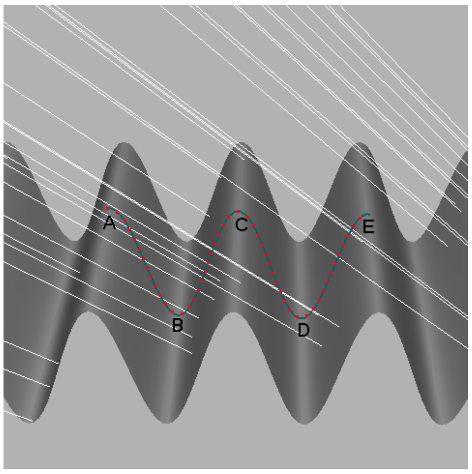
Figure 7.3: *Simple test scenes to study the bias of density estimation near a corner. The test scenes with the ray-photon distribution (white lines) are shown on the left and the 2D density estimates along the depicted red-dotted line for various kernel density estimation methods using the photon map and ray map are shown in the right graphs. In order to interpret the images correctly, note that in the top figure point A is closer to the observer than point B, whereas in the other figures point B is closer.*











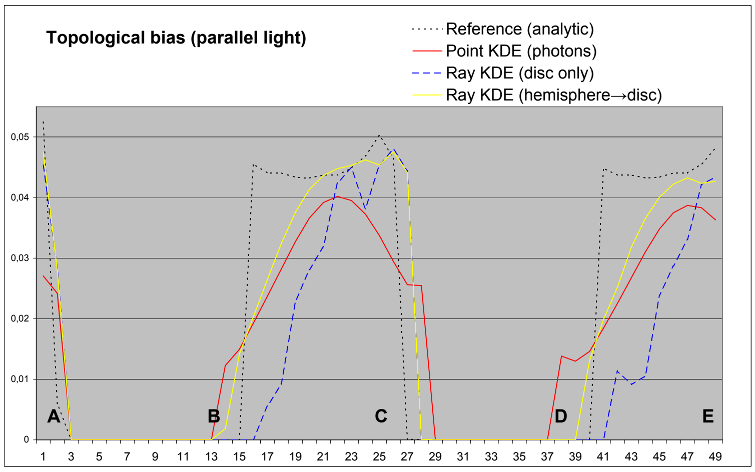


Figure 7.4: *Test scenes that are difficult for density estimation. The scene on the top consists of many tiny surfaces and exhibits strong boundary bias for the photon map. Note how the density estimation from the photon map always underestimates the density since it includes points from many patches even if a small number of nearest neighbors is chosen (here 50). For this case of unconnected patches even the convex hull cannot help! The scenes on the bottom exhibit topological bias which cannot be removed but reduced when using the ray map.*

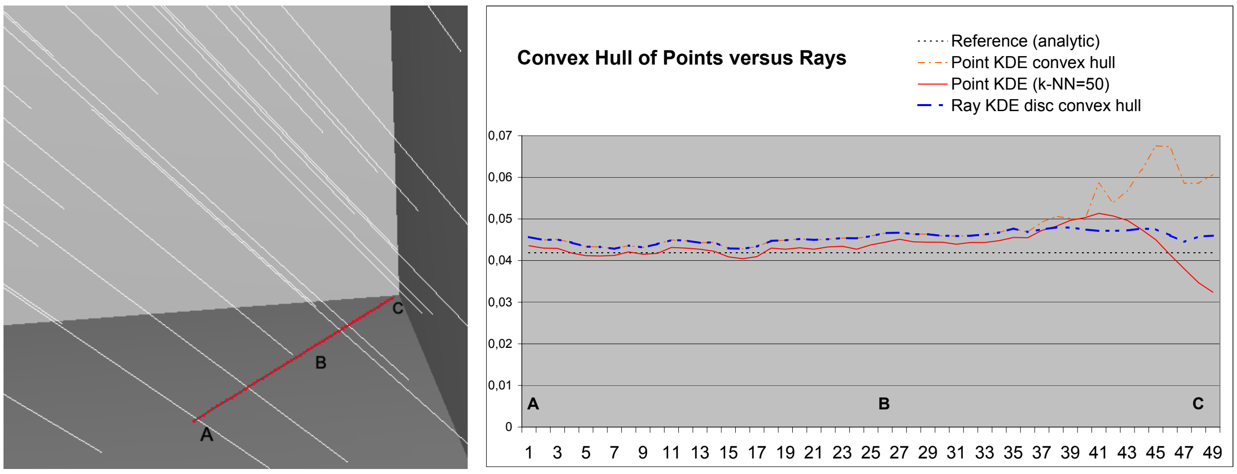


Figure 7.5: *Density estimates using the convex hull area estimate along the depicted line (left) for the photon map KDE and ray map KDE with the disc intersection method. For comparison a standard kNN point density estimation is added (red line). A parallel ray distribution is used with direction shown on the left (white lines). The density is uniform for each plane. Note that the convex hull estimate always overestimates.*

ray-disc intersection domain. All three methods use a simple box kernel for density estimation. Notice that, unless one uses the uniform box kernel for the convex hull, a 2D kernel function must be re-normalized since it does not necessarily integrate to one and is not symmetric anymore. This is because if we use a non-uniform symmetric kernel function and we cut away the area A' from the disc ($A = \pi r^2$) outside the convex hull, we reduce the area by the ratio A'/A. However, the volume of the kernel does not necessarily decrease with the same ratio and can become greater than 1. The result can be observed in the bottom Figure 7.3 for the ray disc convex hull. Using the Epanechnikov kernel it increases incorrectly near the corner. This is in particular problematic for kernel functions which have most of the mass in their center such as the Gaussian or the Biweight kernel function. Normalizing the resulting non-symmetric kernels might be difficult in the general case since the convex hull can have any convex shape. The easiest solution is to use the simplest kernel, the uniform function (box) which gives a correct estimate as shown in Figure 7.5.

7.5 Comparison with the Convex Hull of Photons

For comparison purposes we have also implemented a boundary bias reduction technique for the photon map using the convex hull of the photon hits as an area approximation. Density estimation using photon maps with convex hull area estimates are approximately 2 times faster then ray map queries and 1.7 to 2 times slower than normal photon disc area estimates. The convex hull area compensates for boundary bias on convex isolated surfaces that have got enough photon hits. However the convex hull fails for curved surfaces since we are only concerned with the 2D projection onto the local tangent plane at the query position. Hence, the area of non-flat surfaces is strongly under-estimated (see Figure 7.6) which yields in clearly visible bias.

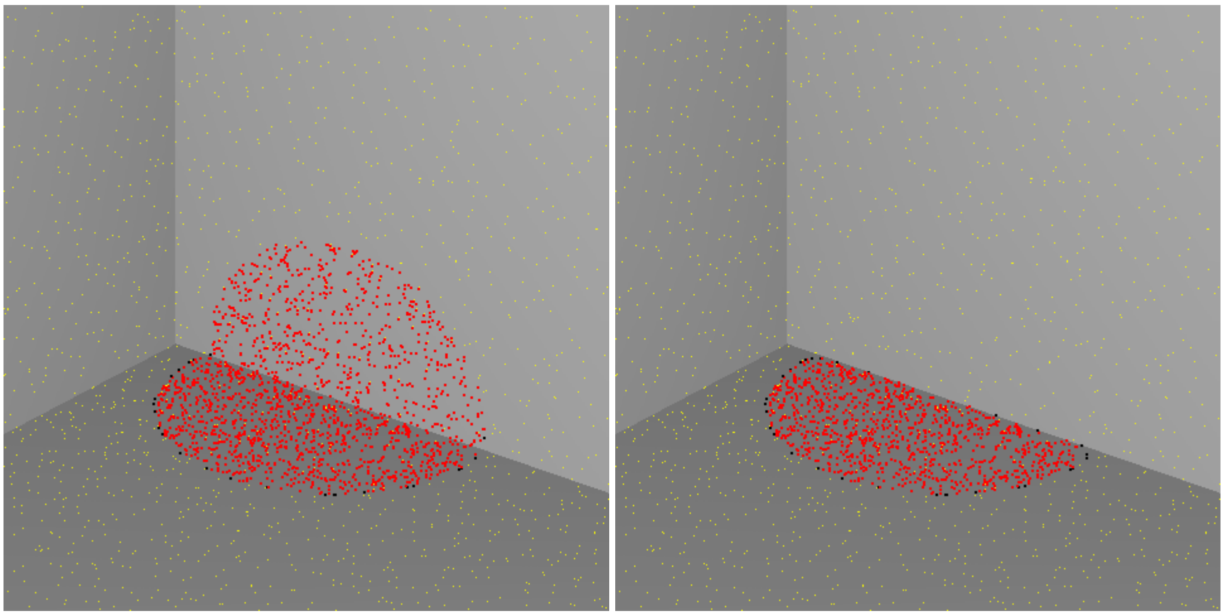


Figure 7.6: *Convex hull area estimate for the photon map (left) and for the ray disc query (right). While the convex hull of the photon map query is problematic for the non-planar query domain, it almost correctly estimates the area for the ray disc intersection domain (right). The black points belong to the convex hull and the red points are included in the density estimate.*

On the other hand, at concave borders, it naturally over-estimates the surface area (Figure 7.7). Another disadvantage of the convex hull area estimate is that it introduces additionally noise even across planar surfaces with constant density due to a slight variance in the surface area estimation which strongly depends on the density of photon hits (Figure 7.7.a/b). And last, the downside of all photon hit density estimation algorithms is the dependence on the size and orientation of the surfaces (see example in top Figure 7.4). Thus, the convex hull area estimates cannot be robustly computed in places where we do not have enough photon hits in the neighborhood. This is normally the case for small surfaces. The fewer photons are included in the convex hull, the higher is the noise in the estimate due to the area underestimation. As an alternative one could compute a 2D spline curve going through the convex hull vertices. This would reduce the bias arising from the polygonal area approximation. However, it is expensive to compute and even more complex to integrate to obtain an area estimate. Another idea for reducing the topological bias is to find an appropriate 3D convex hull at non-planar regions of the 3D model. After all, such an algorithm would be more unstable than a 2D convex hull and also very complex to compute.

7.6 Managing Ray Maps

In this section we describe our implementation of the ray map concept. We present an overview of the method and describe the algorithm for ray map construction, ray map queries, and ray map updates. Developing efficient data structures for ray maps is inversely related to the well-known ray shooting problem and we cannot easily adapt standard acceleration structures used for ray shooting. To our best knowledge and investigation it has not being solved by computational geometry or in other fields dealing with spatial data structures. The proximity

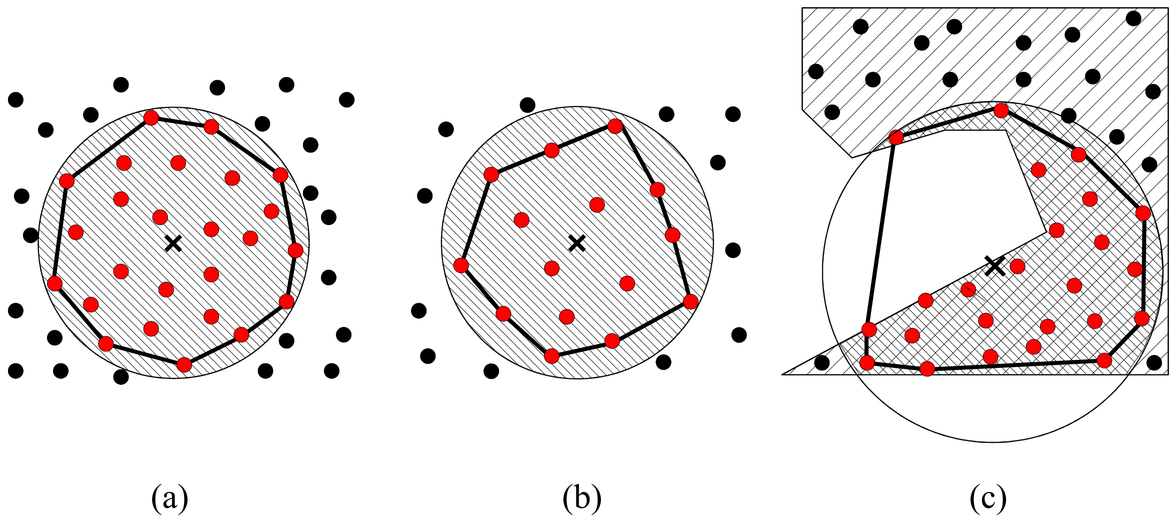


Figure 7.7: *Problems with the convex hull area estimate: the estimated area decreases with the density of photons (a) to (b) and the density estimate is overestimated. If the domain of the photon hits is concave (c) (or unconnected), the area is overestimated and hence the photon density is underestimated.*

search is a 4D problem unlike photon maps, since the lines are 4D primitives. The 4-dimensional search problem increases the running time and memory size of the data structure compared to photon maps. We address the problem by enforcing the coherence of queries, which allows us to achieve reasonable performance while keeping the memory requirements acceptable.

7.6.1 Overview of the Ray Map Data Structure

We represent the ray map using spatial subdivision based on kd-trees. The rays are organized in a way similarly to organizing objects for ray shooting acceleration [27]. We construct a hierarchical spatial subdivision that contains references to rays intersecting the cells of the subdivision. For each elementary cell of the subdivision we maintain a list of references to rays that intersect the cell. Then, for a given query domain, we first determine which cells of the subdivision the domain overlaps and evaluate intersections only with rays referenced in these cells. The resulting spatial subdivision should address the following two points during computation:

- Distribution of rays and queries
- Coherence of the queries

To address the first point we use a termination criteria that stops the hierarchical subdivision based on the number of rays contained in the cells. Additionally, as we describe later, we use a lazy construction of the tree that automatically adapts to the distribution of the queries.

The second point is addressed as follows: once a cell with a set of rays is split, the resulting ray classification is reused by all subsequent queries. If the queries are coherent, only a small number of spatial splits is performed for each query, since we mostly access cells already subdivided by the previous queries.

The proposed strategy using kd-trees that organize rays differs from the case of a kd-tree for ray shooting or point location due to the increased dimensionality of the problem. Our strategy is to keep searching time as low as possible while slightly increasing memory requirements. The increased memory requirements are addressed by enforcing the queries to be coherent in the application. This, together with an efficient caching strategy, allows us to limit the memory usage to an acceptable level.

7.6.2 Static kd-tree Construction

The static construction of the ray map kd-tree proceeds as follows: starting at the root of the tree we check if it satisfies the subdivision criteria. If the subdivision criteria are met, we subdivide the node and distribute all rays it contains to the new leaves. A ray gets associated with a leaf only if it intersects the cell corresponding to the leaf. After the subdivision we continue by recursively traversing the newly created children.

We have used three subdivision criteria: the node is subdivided if all of the following three conditions hold:

1. The number of ray references in the node is greater than a predefined constant C_{min} (we use $C_{min} = 32$).
2. The diagonal of the corresponding cell is longer than a fraction of the diagonal of the scene bounding box R_{min} (typically $R_{min} = 0.1\%$ of the size of the scene bounding box).
3. The depth of the node is smaller than a predefined maximal depth D_{max} (we use $D_{max} = 30$).

We have used splitting planes positioned at the spatial median of the current node that is perpendicular to the axis with greatest spatial extent. The resulting algorithm has several desirable properties:

- Since we use a spatial median the tree is spatially balanced.
- Due to the later presented lazy construction the tree automatically adapts to the query distribution.
- Except for the termination criteria the subdivision is independent of the actual distribution of rays. While this might be a drawback for a static set of rays, it turns out to be beneficial for a dynamically changing ray set and the caching strategy we use.
- We do not have to evaluate a cost function which is required for more advanced splitting plane selection. In asymptotic complexity bounds this reduces the cost of the plane selection from $\mathcal{O}(n \log n)$ (sorting according to the cost) to $\mathcal{O}(n)$. Additionally the constants hidden by the O-notation for the spatial median split are several times lower than for the cost based one, which is important for "on-the-fly" construction.

7.6.3 Intersection Query

An intersection of a given spatial domain with the rays from the ray map is carried out by a constrained traversal of the kd-tree and computing intersections with rays stored at the leaf nodes. The traversal is constrained only to those nodes intersecting the spatial domain of the query. In fact we constrain the traversal to a bounding box of the spatial domain and use the actual domain (disc, sphere, hemisphere) only for evaluating the ray-domain intersection.

Starting at the root node the intersection query proceeds as follows: for each internal node we determine the position of the query bounding box with respect to the plane associated with the node. If the box lies on the positive side of the plane, we continue in the right subtree. Similarly, if the box lies in the negative side of the plane, we continue only in the left subtree. If the box intersects the plane, we recursively continue in both subtrees of the node. Reaching a leaf we compute intersections of the rays associated with the query domain and aggregate all intersecting rays. To avoid testing one ray several times we use mailboxes: once a ray has been tested for intersection, we mark it as tested for the given query.

7.6.4 K-Nearest Neighbors Query

K-nearest neighbors queries aim to locate k nearest rays for the given query center. It uses a similar mechanism as the intersection query, however it requires that the leaf nodes are processed according to their distance from the center of the query. This is achieved by using a priority queue in which the priority of the node is inversely proportional to its distance. This approach is similar to the k-nearest neighbors over point data [3].

Initially we push the root node in the priority queue and proceed as follows: we pop the node with the highest priority from the queue. If it is an internal node, we compute minimal distances d_l, d_r of its children from the query center and insert them in the priority queue with priorities equal to $-d_l$ and $-d_r$, respectively. When reaching a leaf node we evaluate the distance of all rays associated with the node with respect to the query center and add these rays to the ray candidate list. If the ray candidate list becomes larger than k, we apply the k-median algorithm to select the k rays with minimal distance. If the distance of the k-th selected rays is smaller than the distance of the unprocessed node on the priority queue, we can terminate the algorithm, since no unprocessed ray can be closer than the already found k-th ray.

The described technique considers the whole scene as a query domain. It is advantageous to constrain the query domain even for the k-nearest neighbors queries. This is easily incorporated in the algorithm by pushing only those nodes in the priority queue that intersect the query domain. This limits the number of nodes in the queue and thus provides a minor speedup. The set of leaves traversed is mostly identical to the unrestricted query. Note that the priority queue provides a natural adaptation of the traversed part of the scene to the range where the k-nearest rays are actually found, without the need of any complicated estimation techniques. The nodes further away from the k-nearest neighbors are accessed at the higher levels of the hierarchy but they are not accessed any further. The process of the k-nearest neighbors query without and with the restriction of the query domain is illustrated in Figure 7.8.

7.6.5 Dynamic Updates

Inserting a ray into the ray map is easily implemented by a technique similar to ray shooting in kd-trees [27]. We traverse only the part of the kd-tree that is formed by nodes that the given ray intersects. When reaching a leaf node we insert a reference to the ray. Optionally we could apply some rebalancing techniques after inserting a number of non-evenly distributed rays. However as we shall see in Section 7.7.1 our lazy ray map construction postpones the rebalancing to the moment of the actual presence of a query in this particular part of the scene.

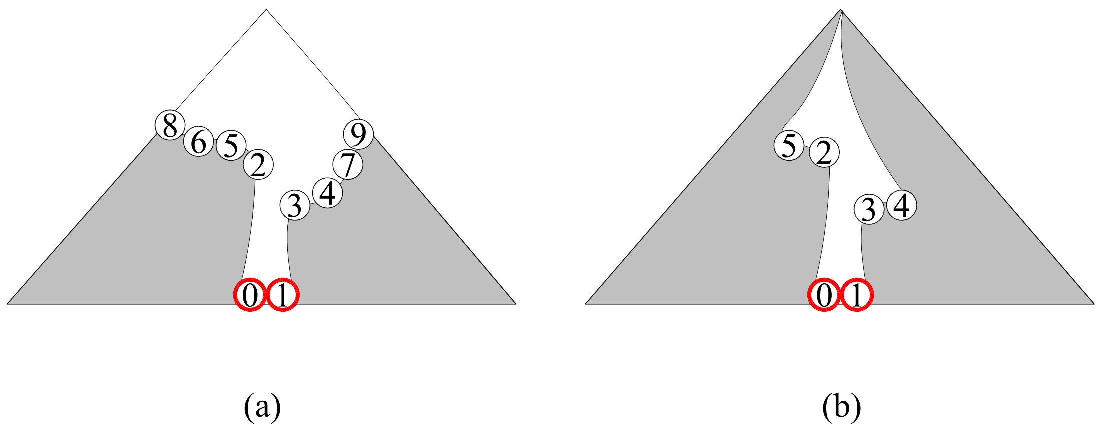


Figure 7.8: *K-nearest neighbors queries using a priority queue. (a) Traversal of the tree for the unconstrained query, i.e. query with a domain corresponding to the whole scene. The nodes are labeled according to their processing order. The rays at the red leaves were actually tested for intersection. After testing these two nodes the query was terminated since we have found a required number of rays that were closer than any of the unprocessed nodes in the queue. (b) Traversal of the tree constrained by the query domain. Note that although we have accessed a smaller part of the hierarchy we actually test the same number of leaves as for the unconstrained query.*

Deleting a ray from the ray map proceeds similarly as the ray insertion. Again we perform a traversal of the tree constrained to the nodes intersecting the ray. When reaching a leaf we remove the reference to this ray from its ray list. Deleting rays one by one could suffer from exhaustive searching in the leaf ray lists. One way to tackle this problem is to keep the ray list sorted and use a binary search to locate the actual ray reference. However this requires resorting the list after each modification. A better strategy is to aggregate the deletions and perform deletion using a number of rays in a single pass through the list as follows: Each ray keeps the number of references to itself. All rays to be deleted are marked. We perform the ray traversal starting with the first ray in the list. Reaching a leaf node we scan the associated list of rays and remove *all* rays scheduled for deletion. In addition, this leaf is then denoted as solved. We continue with the next ray by first checking if it still has a non-zero reference count. If the reference count equals to zero, this ray was already deleted by previous ray traversals. Additionally, we can skip the processing of leaves that were previously marked as solved. This technique significantly improves the performance for deleting coherent rays that are referenced in similar sets of leaves.

7.7 Ray Map Enhancements

This section presents several enhancements of the kd-tree based ray map implementation. The first three presented methods aim to improve the performance of queries. The last method limits the size of the ray map allowing the user to trade the total memory consumption for speed.

7.7.1 Lazy Ray Map Construction

In order to concentrate the splits in areas really accessed by the queries we use a lazy kd-tree construction. The kd-tree is constructed by interleaving the traversal of the already existing part of the tree with subdivisions performed on its leaf nodes that satisfy the subdivision criteria (e.g. contain too many ray references).

Given a query with its domain corresponding to an axis aligned box, we start at the root node and proceed as follows:

- If the current node is an interior node, we check the position of the box with respect to the node's associated plane and continue the traversal recursively for the subtrees intersected by the box.
- If the current node is a leaf, we check if the subdivision criteria are met (Section 7.6.2):
  - If the subdivision criteria *are met*, we subdivide the node and distribute all ray references it contains to the new leaves. Recall that a ray is associated with a leaf only if it intersects the corresponding cell. After the subdivision we continue with the traversal of the newly created children.
  - If the subdivision criteria *are not met*, we test all rays associated with the node for an intersection with the query box.

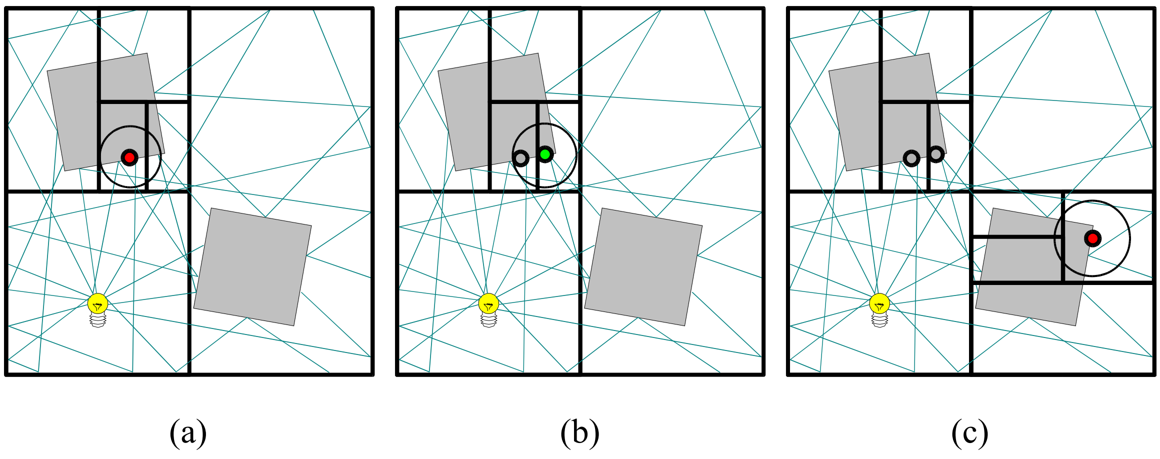


Figure 7.9: *Lazy construction of the kd-tree driven by the queries. (a) The first query depicted by the red disc requires five subdivisions of the kd-tree till the required limit of 5 rays per node is fulfilled in its kernel support. (b) The second query (green disc) does not require any new subdivision due to its spatial coherence with the previous query. (c) The third query subdivides the kd-tree further and adds four new internal nodes.*

7.7.2 Directional Splits

Two important ray map queries use a query domain that does not only restrict the spatial range of the rays but also their directional range. In particular the disc query and the hemisphere query only consider rays with a negative dot product with the normal of the disc or the hemisphere. If we only group rays according to the spatial positions, we cannot efficiently cull groups of rays that do not contribute to the query because their direction is opposite to the reference direction.

To tackle this problem we extended the kd-tree by *directional nodes*. Unlike the usual kd-tree node the directional node does not provide a split in the spatial domain, but rather in the directional domain. The directional node contains a reference direction. The node subdivides the current range of directions into those having a positive and negative dot product with the reference direction. For the density estimation the negative directions are those feasible for the disc or hemisphere intersection queries since they represent incoming rays. To allow sharing of the same directional node by several queries with slightly different normals we enlarge the set of feasible directions by a specified α angle (see Figure 7.10). Such an α-extended directional node does not cull all infeasible rays, but it allows to reuse the directional node by all queries with normals within the α threshold from the reference direction. In scenes with directionally coherent queries, we have observed that using $\alpha = 10$ degrees, about 90% of all query normals were within the α range. This means that we could successfully cull the whole subtree maintaining rays within the remaining $180 - 2\alpha$ degrees. Moreover, since the incoming ray direction is mainly cosine distributed for diffuse scenes, there are usually fewer rays than $2\alpha/180$ of all infeasible rays arriving at a grazing angle from the back side of the query domain. Hence, the major part of infeasible rays is culled efficiently.

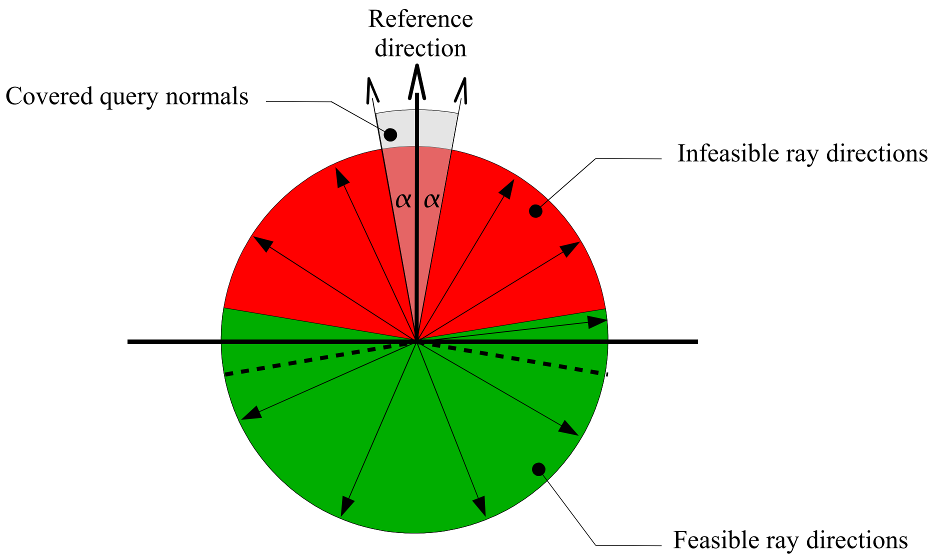


Figure 7.10: *The subdivision of the ray directions according to a directional node.*

In the optimal case we would place the directional nodes as high in the tree as possible while making sure that the whole subtree corresponds to a spatial domain where directionally coherent queries are expected. We use a simplified strategy suitable for dynamic tree construction that uses two predefined constants: the minimal depth of a node and minimal diagonal size of the cell. If we shall subdivide a node during the lazy tree construction and these two criteria are met for the given node, we first check if there was no other directional node on the path to the root that covers the given query (i.e. the angle between its reference direction and the query normal $< \alpha$). If we do not find such a node we introduce the directional node and split the current set of rays according to the angle between their directions and the reference direction as shown in Figure 7.10.

7.7.3 Exploiting Query Coherence

Subsequent queries in the ray map are likely to be coherent, if they are induced by primary rays in the direct visualization of the density estimation using a coherent pixel order in the image. We exploit query coherence by reducing repeated traversals of the same interior nodes of the kd-tree. The tests at the leaf nodes are carried out as usual since for most applications we need to evaluate the actual ray distances for each particular query.

Our design goal was to provide a mechanism that does not require preprocessing of the queries but allows us to use the coherence between subsequent queries if there is some. We only describe a modification to the k-nearest neighbors algorithm. The modification of the fixed domain intersection algorithm works similarly.

For the first query we create a list of nodes corresponding to the reached leaves and unprocessed nodes on the priority queue that are within an ϵ-distance from the k-th found ray. The created list becomes a reference list for subsequent queries and the query center becomes the reference center. For the next query we first check if the query center lies within the ϵ-distance from the reference center. If this is the case, we push all the nodes from the reference list to the priority queue using the actual priorities with respect to the new query center. The query then proceeds as usual, but the reference list and the reference center are not modified. If the query center does not lie within the ϵ-distance from the reference center, we start the traversal at the root node and create a new reference list that will possibly be used by subsequent queries.

7.7.4 Limiting Memory Usage

The existing data structures dealing with ray space share a common problem of high storage costs due to the 4-dimensionality of the search problem. As a result the number of rays stored in the ray map can be very large. The rays stored in the ray map are generated by the actual global illumination algorithm. This algorithm can employ numerous strategies for decreasing the number rays while keeping high accuracy of the illumination representation, such as importance sampling or energy redistribution among existing samples for controlling the sample density [77]. These techniques are independent of the ray map concept as long as they only require evaluation of ray proximity queries possibly followed by a ray map update. Even when these techniques are applied, there can still be a huge number of rays stored in the ray map. The ray map implementation should however be able to limit the size of the indexing structure and so to balance the query performance and memory costs.

Our ray map implementation stores multiple references to a ray in several leaves of the kd-tree. As the kd-tree is constructed lazily, the overall memory consumption can grow with the number of processed queries. The actual growth rate depends on the distribution of the rays (mainly their length and direction) and the spatial coverage of the queries. If the rays are very short and the queries cover only a small fraction of the scene the memory growth will be small. On the other hand for long rays and queries evenly filling the space, there might be many leaves with references to each ray in the cells of the constructed subdivision which increases the total memory cost.

We have developed a mechanism allowing to efficiently balance the memory dedicated to the

Scene	Rays $[10^3]$	Queries $[10^3]$	Method	Found Rays	Successful Tests [%]	Memory [MB]	Collapses [%]	Query Time [ms]	Speedup [-]
Cornell Box	1887	53	SP	100	0.42	23.0	-	24.9	1.0
			RM	100	25.0	128.0	0.02	0.88	28.3
Cognac	67	128	SP	78	0.50	2.4	-	3.27	1.0
			RM	78	8.3	3.7	0	0.24	13.6
Office	2550	307	SP*	100	0.9	62.0	-	3.75	1.0
			RM	100	16.9	78.0	0	0.22	17.0
Sala	2360	1310	SP	100	0.40	33.0	-	0.89	1.0
			RM	100	19.60	128.0	0.0001	0.20	4.5

Table 7.1: *Comparison of the k-nearest neighbors query performance for the kd-tree based ray map implementation and the dynamic list of spheres. *For the last SP test we had to reduce the search radius to 0.5% of the scene size to obtain reasonable timings. When the initial radius was larger there were too many rays in the candidate list leading to running times of more than two orders of magnitude greater than the ray map method.*

ray map and the computational cost using the least-recently-used (LRU) caching strategy. We set a limit on the memory usage for representation of the kd-tree, such as 100 MBytes. Before each subdivision of a node in the kd-tree, we check if the limit has not been exceeded. If it has been exceeded, we search for the LRU subtree of the kd-tree and collapse it to a single node. The collapses of subtrees are performed until the desired memory bound is reached. Then the required subdivision of the current node is performed.

The described method maintains parts of the kd-tree that were recently accessed. In this way, we make sure that the memory usage will not exceed a predefined memory limit, while we can still exploit coherence of subsequent queries.

7.8 Results

In this section we summarize the results obtained using our implementation of ray maps in the context of density estimation. We compare the achieved results for the direct visualization using ray maps with the direct visualization using photon maps.

We have conducted four different tests. The first test compares k-nearest neighbors queries using our ray map implementation and the ray cache using the dynamic list of spheres [47]. The second test illustrates the dependence of the query performance on the number of rays stored in the ray map. The third test evaluates the performance of the queries in dependence on the number of requested nearest rays. The fourth test compares the performance of density estimation from photon maps and ray maps for the direct visualization.

The first test is the comparison of ray maps with the ray cache. We conducted tests on four different scenes: the Cornell box, the Cognac, the Office, and the Sala scene (see Figure 7.12). For each test we have measured the total number of rays, the number of queries, the number of actually found rays, the percentage of successfully tested rays, the peak memory used during the rendering, the percentage of sub-tree collapses per query (enforced by the caching scheme), and the average query time. We have used k-nearest neighbors queries which were set to find 100 nearest rays. Additionally, we have restricted the search to a distance corresponding to 5% of the radius of the scene's bounding sphere. The results are summarized in Table 7.1.

We can see that the ray map method provides significant speedup compared to the ray-cache [47]. From the running statistics we figured out two main reasons:

Rays 10^3	Found	Succ. Tests [%]	Query Time [ms]
189	100	27.0	0.15
378	100	27.7	0.20
944	100	26.0	0.40
1887	100	25.0	0.77

Table 7.2: *Dependence of the query performance on the number of rays stored in the ray map for the Cornell box scene. The results were averaged using 53,000 nearest neighbor queries.*

Found	Succ. Tests [%]	Query Time [ms]
20	11.5	0.63
50	19.2	0.68
100	25.0	0.77
200	33.4	0.89
500	44.0	1.29

Table 7.3: *Dependence of the query performance on the number of requested nearest neighbors. The measurements were carried out for the Cornell Box using* 1.8×10^6 *rays and 53,000 queries.*

1. the ray-cache using the dynamic list of spheres heavily depends on the choice of the appropriate search radius. If this radius is larger than that of the actual neighborhood, where the k rays are found we obtain too many candidate rays that have to be ranked.

2. In case that all subsequent queries are not coherent enough, the dynamic list of spheres has to be reconstructed which is relatively costly.

The second test shows the dependence of the query performance on the total number of rays stored in the ray map. This test was carried out for the Cornell box using 53,000 nearest neighbor queries. The results are summarized in Table 7.2. We can observe that increasing the number of rays causes only a sub-linear increase of the average time per query. However, the query time has no logarithmic complexity which is due to the fact that the average query time also includes the costs for the lazy construction of the kd-tree.

The third test depicts the dependence on the number of requested rays for a k-nearest neighbors query. Again we have used the Cornell box with 1.8×10^6 rays and 53,000 queries.

We also see a sub-linear increase of query time when increasing the number of desired rays. The more rays we require the farther we have to search from the query center. An important property of our ray map implementation is that due to the priority queue based traversal it automatically establishes the neighborhood where the desired number of rays are being found without significant performance lost.

The fourth test compares the rendering using the direct visualization with ray maps and photon maps. The time performance results for this test are summarized in Table 7.4, the images are shown in Figure 7.11 for the photon map and in Figure 7.12 for the ray map. Notice the clearly visible boundary bias for the photon map.

The comparisons have demonstrated that the ray map based density estimation successfully eliminates the boundary bias. The overhead of performing the ray map queries is moderate – the direct visualization (i.e. density estimation) for the ray map was 3.9 to 5.2 times slower than the direct visualization for the photon map.

Scene	Time [s] Ray map	Time [s] Photon map	Ratio [-]
Cornell Box	90.0	23.0	3.9
Cognac	104.0	20.0	5.2
Office	86.6	20.0	4.3
Sala	78.2	19.6	4.0

Table 7.4: *Rendering times for density estimation with the photon map and the ray map without final gathering for a resolution of* 400×400 *pixels.*

For all the tests, we have set the limit for the memory usage for the ray map as described in Section 7.7.4 to 128 MBytes.

7.9 Discussion

The experimental evaluation of our ray map implementation revealed several interesting features of the method. For the discussion we select the splitting plane selection and a comparison to a different ray map implementation that uses a dual space.

7.9.1 Splitting Plane Selection

Inspired by the rich literature on kd-trees for ray shooting [39, 27] we have experimented with other methods for the splitting plane selection such as the ray median or query distribution heuristics. The ray median selects a splitting plane so that the number of rays in the left and right subtrees of the split node is equal (plus/minus one ray). This strategy results in a balanced tree with least maximal depth. The query distribution heuristics is based on a similar idea as the surface area heuristic subdivision for ray shooting. We estimate the costs of a splitting plane position by weighting the number of rays in the left and right child by the probability that the corresponding child will be accessed by a query.

Surprisingly, the conducted experiments have shown that the best overall query performance was achieved by using the simplest strategy - the spatial median split discussed in Section 7.6.2. When using the more advanced ray median split or query distribution heuristic we obtained about 10-30% performance slowdown per query. We explain this result as follows:

- The computational cost of the splitting plane selection for the advanced techniques is higher. Asymptotically this means $\mathcal{O}(n \log n)$ versus $\mathcal{O}(n)$, but there is also an additional cost for evaluating the heuristic function hidden in the O-notation. Since we construct the tree lazily and the number of queries is comparable or even lower than the number of rays stored in the ray map this difference becomes significant.

- The heuristics should provide a well-balanced tree with respect to the queries. However we deal with a more complicated higher-dimensional problem than for traditional kd-trees that store only points in 3D. Any ray can be referenced in a number of leaves and it is hard to predict how many references will occur at each subtree when performing a split near the root of the tree. This is emphasized by the fact that long diagonal rays can span across the whole scene, although after the subdivision they end up in only a few nodes in proximity of the ray.

7.9.2 Ray Maps in Line Space

A natural candidate for the ray map representation is the line space. Rays on a given line in primary space are represented by a point in line space. By representing all rays as line space points we can cluster these points and use classical range searching methods to find points that intersect the line space mapping of the query domain. In fact this technique was used in our first ray map implementation. We have used Plücker coordinates to map supporting lines of the rays to 6D points (points in 5D projective space embedded in 6D). Then we have clustered the resulting points using a balanced BBD-tree [3]. The query domain (disc) that was mapped to a line space convex polyhedron formed a set of 6D hyperplanes [81]. We have then found all points of the BBD-tree that were contained in the polyhedron. To reflect the fact that we actually deal with line segments instead of lines we have interleaved the search with testing intersections of the query domain with the spatial bounding boxes of ray clusters. Unfortunately, the resulting technique exhibits rather small performance gain compared to the naive implementation. The major problem is performing a computationally efficient intersection of the line space mapping of the query domain (6D unbounded convex polyhedron) and the 6D bounding boxes corresponding to clusters of rays. The query domain is compact in primal space (e.g. disc), but after mapping to line space we typically obtain an unbounded thin polyhedron. Thus it is not possible to efficiently approximate the polyhedron with a box and we have to perform intersection of the polyhedron itself with the boxes of the BBD-tree.

To avoid complicated boundary intersection test we have used a conservative test that is based on finding a separation plane. As candidates for separation planes we used all boundary faces of the polyhedron. If the box is on the negative side of the plane it cannot intersect the polyhedron. Unfortunately we have observed that such a test only succeeds at the bottom of the hierarchy and does not cull of larger ray clusters near the root. As a result the BBD-tree was always traversed almost to the bottom providing only a ten fold speedup compared to the naive implementation of the query.

This approach however devotes further investigation since keeping rays as points in a dual space has the advantage of easily predictable memory costs: every ray is represented by exactly one entry. Perhaps a combination of primal/dual space data structure could share the benefits of both: compactness of the query domain (primal space) and elegance of the ray representation (dual space).

7.10 Future Work

The ray map concept opens a number of topics for future work. First, we would like to work on the methods detecting or reducing occlusion bias via simple statistical means over the rays in query. Second, it is possible to use the ray map for rendering of volumetric effects using 3D density estimation. An extension of the algorithm to 3D density estimation is very straightforward and neither needs any modification (except for the used kernel) of the ray map nor would it need additional memory unlike the photon map. Third, the ray map is mainly intended to be used exclusively for the direct visualization of photon density estimation. Therefore, we would like to combine the ray map with more sophisticated bandwidth selection in order to detect large illumination gradients at shadows boundaries and caustics. And finally, we envision a composite rendering algorithm that computes the indirect illumination more efficiently via a combination of density estimation from the photon map, the ray map, and the final gathering across the image plane. When revising the underlying assumptions for each method, starting with the cheapest method first, a significant speedup for rendering could be achieved without a decrease of image quality compared to costly final gathering over the whole image. Last but not least, we would like to use the ray map in the context of animation, without the necessity to compute the ray map from the scratch for each frame.

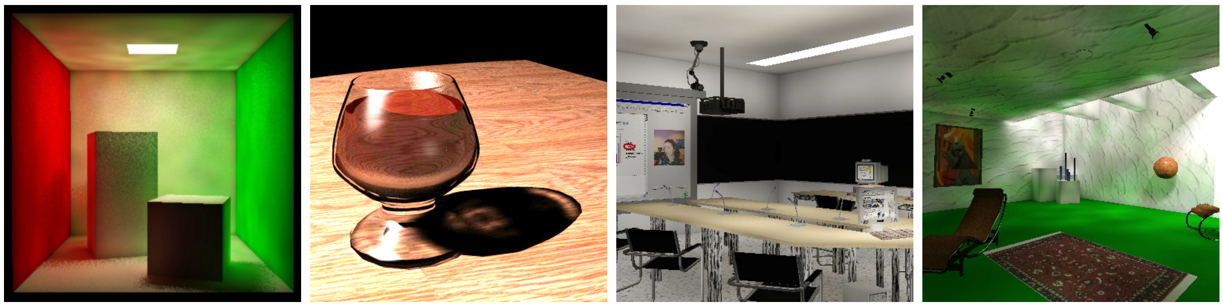


Figure 7.11: *The test scenes rendered using photon tracing with the direct visualization of the photon map using density estimation: the Cornell box, Cognac, Office, and Sala.*

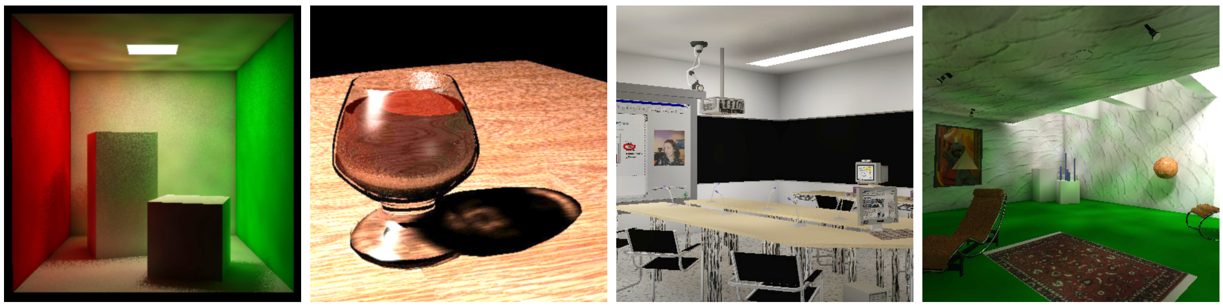


Figure 7.12: *The test scenes rendered using photon tracing with the direct visualization of the ray maps using density estimation: the Cornell box, Cognac, Office, and Sala.*

Chapter 8

Conclusion

We have described the fundamentals of global illumination and density estimation at the beginning of the thesis. We described the classic photon mapping method and the involved difficulties arising from density estimation. In addition to the standard algorithm, several enhancements were discussed including irradiance caching, final gathering, and importance sampling. We proposed a novel method to speed up the k-nearest neighbors search in the photon tree by computing an adaptive initial search radius using the spatial kd-tree.

We continued with a survey of search data structures that are suitable for photon mapping and proposed an efficient spatial kd-tree with small memory footprint. A full comparison of the described data structures was given at the end of the chapter.

We presented two practical applications of density estimation in photon mapping to compute a global illumination solution for arbitrary lighting situations and scenes. The first application, *reverse photon mapping*, trades memory consumption with rendering speed and highly improves the coherence in the density estimation. We achieved this by utilizing a dual-tree approach and reversing the search for the density estimation. Two kd-trees are constructed: a kd-tree over photons and also a kd-tree over final gather ray hit points (reverse photons) which we called *reverse photon map*. The indirect diffuse and glossy illumination is then computed by photon density estimation. Instead of performing a density estimate at each reverse photon location to compute an irradiance estimate, we distribute the energy of each photon to neighboring reverse photons. This is algorithmically superior since the number of photons is much smaller than the number of reverse photons.

We provided a theoretical analysis for the complexity of the reverse photon mapping algorithm compared to normal photon mapping. Further, several optimizations were presented to increase the utility of the algorithm: the modified irradiance caching (Section 6.7.3), the final gather ray shooting cache (Section 6.9.1), which is also applicable to other ray shooting algorithms, the aggregate search in the dual-tree (Section 6.9.2), and the offline density control for the photon map (Section 6.9.3).

Furthermore, we showed successfully how to deal with the enormous memory overhead via screen tiling and the use of external data structures. We evaluated our method for various test scenes of different complexity and compared it with the normal photon mapping algorithm [34]. We have also demonstrated that the method is also capable of rendering global illumination images in a setting as it is used for production rendering.

In the next chapter, we presented a data structure for representing light transport called *ray* *map*. The ray map extends the concept of the photon map: it provides a general mechanism for storing light path samples as well as retrieving the samples using ray proximity queries. We have discussed the intersection queries, nearest neighbor queries, and practically analysed their combinations for a variety of test cases with our density estimation framework. The ray map does not only allow to determine rays in proximity, but also allows to use new distance metrics unavailable for the photon map.

We described an implementation of the ray map based on a kd-tree. To achieve query performance approaching the performance of the photon map while keeping memory requirements low, we proposed a number of techniques. The kd-tree is constructed lazily based on the actual queries, it is extended by directional nodes for efficient culling of infeasible rays for the queries with limited directional range, and we exploit query coherence by avoiding repeated traversals in the upper part of the tree. Finally, a method was described for limiting the memory usage of the ray map by caching only the part of the tree that was recently accessed.

We practically evaluated the algorithm by direct visualization of the ray map and showed that we can avoid boundary bias inherent to photon maps. By searching with a hemispherical domain, while computing the surface area on the disc, we also reduced the topological bias. The results were achieved at the cost of moderate increase in computation time compared to photon maps.

Bibliography

[1] I. S. Abramson. On Bandwidth Variation in Kernel Estimates - a Square Root Law. In *The Annals of Statistics*, volume 10, pages 1217–1223, 1982.

[2] S. Ar, G. Montag, and A. Tal. Deferred, Self-Organizing BSP Trees. *Computer Graphics Journal (Eurographics '02)*, 21(3):C269–C278, 2002.

[3] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An Optimal Algorithm for Approximate Nearest Neighbor Searching Fixed Dimensions. *Journal of the ACM*, 45(6):891–923, 1998.

[4] J. L. Bentley. Multidimensional Binary Search Trees in Database Applications. In *IEEE Trans. on Soft. Eng.*, pages 333–340, 1979.

[5] L. Breiman, W. Meisel, and E. Purcell. Variable Kernel Estimates of Multivariate Densities. In *Technometrics*, volume 19, pages 135–144, 1977.

[6] D. Burke, A. Ghosh, and W. Heidrich. Bidirectional Importance Sampling for Direct Illumination. In *Rendering Techniques 2005*, pages 147 – 156. Eurographics Symposium on Rendering, 2005.

[7] M. Cammarano and H. W. Jensen. Time Dependent Photon Mapping. In *Rendering Techniques 2002*, pages 135–144, June 2002.

[8] P. H. Christensen. Faster Photon Map Global Illumination. In *Journal of Graphics Tools*, volume 4, pages 1–10, 1999.

[9] P. H. Christensen. Photon Mapping Tricks. *SIGGRAPH Course Notes*, 43:93–121, 2002.

[10] P. H. Christensen and D. Batali. An Irradiance Atlas for Global Illumination in Complex Production Scenes. In *Rendering Techniques 2004*, pages 133–141. Proceedings of Eurographics Symposium on Rendering, 2004.

[11] M. Cohen and D. Greenberg. The Hemi-Cube, a Radiosity Solution for Complex Environments. In *Computer Graphics*, pages 31–40. Proceedings of SIGGRAPH '85, 1985.

[12] M. F. Cohen and J. R. Wallace. *Radiosity and Realistic Image Synthesis.* Academic Press, San Diego, CA, 1993.

[13] R. Cook, T. Porter, and L. Carpenter. Distributed Ray Tracing. In *Computer Graphics*, pages 137–145. Proceedings of SIGGRAPH '84, 1984.

[14] M. Dickeerson, C. Duncan, and M. Goodrich. Kd-trees are better when cut at the longest side. In *Proceedings of the 8th Annual European Symposium on Algorithms*, pages 179–190, 2000.

[15] J.-M. Dischler. Efficiently Rendering Macro Geometric Surface Structures with Bi-Directional Texture Functions. In *Rendering Techniques '98*, pages 169–180, 1998.

[16] G. Drettakis and F. X. Sillion. Interactive Update of Global Illumination using a Line-Space Hierarchy. In *Computer Graphics*, volume 31, pages 57–64. ACM SIGGRAPH Proceedings, 1997.

[17] P. Dutré, P. Bekaert, and K. Bala. *Advanced Global Illumination.* A K Peters, Natick, MA, 2003.

[18] V. Gaede and O. Günther. Multidimensional Access Methods. *ACM Computing Surveys*, 30(2):170–231, 1998.

[19] P. Gautron, J. Křivánek, K. Bouatoch, and S. Pattanaik. Radiance Cache Splatting: A GPU-Friendly Global Illumination Algorithm. In *Rendering Techniques 2005*, pages 55–64. Eurographics Symposium on Rendering, 2005.

[20] P. Gautron, J. Křivánek, S. Pattanaik, and K. Bouatouch. A Novel Hemispherical Basis for Accurate and Efficient Rendering. In *Rendering Techniques 2004*. Eurographics Symposium on Rendering, 2004.

[21] S. J. Gortler, P. Schroder, M. F. Cohen, and P. Hanrahan. Wavelet Radiosity. In *Computer Graphics*, pages 221–230. Proceedings of SIGGRAPH '89, 1989.

[22] G. Greger, P. Shirley, P. M. Hubbard, and D. P. Greenberg. The Irradiance Volume. *IEEE Comput. Graph. and Appl.*, 18(2):32–43, 1998.

[23] P. Hall and J. S. Marron. Extent to which Least-squares Cross-validation minimizes Integrated Squared Error in Non-Parametric Density Estimation. In *Probability Theory Rel. Fields*, volume 74, pages 567–581, 1987.

[24] J. Hammersley and D. Handscomb. *Monte Carlo Methods.* Chapman and Hall, London, 1964.

[25] P. Hanrahan, D. Salzman, and L. Aupperle. A Rapid Hierarchical Radiosity Algorithm. In *Computer Graphics*, volume 25, pages 197–206. Proceedings of SIGGRAPH '91, 1991.

[26] D. Hansson and N. Harrysson. Fast Photon Mapping using Grids. Master's thesis, 2002.

[27] V. Havran. *Heuristic Ray Shooting Algorithms.* Ph.d. thesis, Czech Technical University in Prague, November 2000.

[28] V. Havran, J. Bittner, R. Herzog, and H.-P. Seidel. Ray Maps for Global Illumination. In *Rendering Techniques 2005*, pages 43–54. Eurographics Symposium on Rendering, 2005.

[29] V. Havran, R. Herzog, and H.-P. Seidel. Fast Final Gathering via Reverse Photon Mapping. In *Computer Graphics Forum*, volume 24, pages 323–333. Proceedings of Eurographics, 2005.

[30] P. Heckbert. Adaptive Radiosity Textures for Bidirectional Ray Tracing. In *ACM SIGGRAPH Proceedings '90*, volume 24, pages 145–154, 1990.

[31] H. Hey and W. Purgathofer. Global Illumination with Photon Mapping Compensation. Technical Report TR-186-2-01-04, Vienna University of Technology, January 2001.

[32] W. Ingo. Photorealistic Rendering using the Photonmap. Master's thesis, 1999. Diplomarbeit, Universität Kaiserslautern.

[33] H. W. Jensen. Global Illumination using Photon Maps. In *Rendering Techniques '96*, pages 21–30. Proceedings of the Seventh Eurographics Workshop on Rendering, 1996.

[34] H. W. Jensen. *Realistic Image Synthesis Using Photon Mapping.* A. K. Peters, Natick, 2001.

[35] H. W. Jensen and N. J. Christensen. Photon Maps in Bidirectional Monte Carlo Ray Tracing of Complex Objects. In *Computers & Graphics*, volume 19, pages 215–224, 1995.

[36] P. Jiménez, F. Thomas, and C. Torras. 3D Collision Detection: A Survey. *Computers and Graphics*, 25(2):269–285, Apr. 2001.

[37] J. T. Kajiya. The Rendering Equation. In *Computer Graphics*, pages 143–150. Proceedings of SIGGRAPH '86, 1986.

[38] M. Kalos and P. Whitlock. *The Monte Carlo Method, Volume 1: Basics.* John Wiley and Sons, New York, 1986.

[39] M. R. Kaplan. The Use of Spatial Coherence in Ray Tracing. In *Techniques for Computer Graphics*, pages 173–193. 1987.

[40] T. Kato, H. Nishiumura, T. Endo, T. Maruyama, J. Saito, and M. Adachi. Photon Mapping in Kilauea. *SIGGRAPH Course Notes*, 43:122–191, 2002.

[41] A. Keller. Quasi-Monte Carlo Radiosity. In *Rendering Techniques '96*, pages 101–110, 1996.

[42] A. Keller. Instant Radiosity. In *Proceedings of SIGGRAPH '97*, pages 49–56, 1997.

[43] A. Keller and I. Wald. Efficient Importance Sampling Techniques for the Photon Map. In *Proc. Vision, Modelling and Visualization*, pages 271–279, 2000.

[44] E. Lafortune and Y. D. Williams. A 5D Tree to Reduce the Variance of Monte Carlo Ray Tracing. In *Rendering Techniques '96*, pages 11–19, 1996.

[45] E. P. Lafortune and Y. D. Willems. Bi-Directional Path Tracing. In *Compugraphics 93*, pages 145–153, 1993.

[46] B. D. Larsen and N. J. Christensen. Optimizing Photon Mapping using Multiple Photon Maps for Irradiance Estimates. In *WSCG Poster Proceedings, Plzen, Czech Republic*, pages 77–80, 2003.

[47] M. Lastra, C. Urena, J. Revelles, and R. Montes. A Particle-Path Based Method for Monte Carlo Density Estimation. In *In Poster Papers Proceeding of the 13th Eurographics Workshop on Rendering*, pages 33–40, June 2002.

[48] S.-K. László. *Monte Carlo Methods in Global Illumination.* Institute of Computer Graphics, Vienna University of Technology, 1999.

[49] F. Lavignotte and M. Paulin. A New Approach of Density Estimation for Global Illumination. In *Proceedings of WSCG 2002*, pages 263–270, 2002.

[50] F. Lavignotte and M. Paulin. Scalable Photon Splatting for Global Illumination. In *Graphite 2003 (International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia), Melbourne, Australia.*, pages 1–11. ACM SIGGRAPH, 2003.

[51] M. C. Lin and S. Gottschalk. Collision Detection between Geometric Models: a Survey. In *Proc. of IMA Conference on Mathematics of Surfaces*, pages 37–56, 1998.

[52] T. MacRobert. *Spherical Harmonics; An Elementary Treatise on Harmonic Functions, with Applications.* Dover Publications, 1948.

[53] S. Maneewongvatana and D. Mount. It's okay to be skinny, if your friends are fat. In *Proceedings of the 4th Annual CGC Workshop on Computational Geometry*, 1999.

[54] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, H. Teller, and E. Teller. Equations of State Calculations by Fast Computing Maschines. *Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[55] D. P. Michell and A. N. Netravali. Reconstruction Filters in Computer Graphics. In *Proceedings of SIGGRAPH '88*, pages 221–228, 1988.

[56] G. Müller, J. Meseth, M. Sattler, R. Sarlette, and R. Klein. Acquisition, Synthesis, and Rendering of Bidirectional Texture Functions. In *Computer Graphics Forum*, pages 83–109, 2005.

[57] H.-G. Müller. Empirical Bandwidth Choice for Nonparametric Kernel Regression by means of Pilot Estimators. In *Statistical Decisions Supplement no. 2*, pages 193–206, 1985.

[58] K. Myszkowski. Lighting Reconstruction using Fast and Adaptive Density Estimation Techniques. In *Eurographics Rendering Workshop 1997*, pages 251–262, 1997.

[59] L. Neumann, A. Neumann, and L. Szirmay-Kalos. Analysis and Pumping up the Albedo Function. Technical Report TR-186-2-98-20, Institute of Computer Graphics, Vienna University of Technology, 1998.

[60] H. Niederreiter. Random Number Generation and Quasi-Monte Carlo Methods. In *CBMS-NSF Regional Conference Series in Appl. Math.*, volume 63, Philadelphia: SIAM, 1992.

[61] S. N. Pattanaik and S. P. Mudur. Computation of Global Illumination by Monte Carlo Simulation of the Particle Model of Light. In *Third Eurographics Workshop on Rendering.*, pages 71–83, 1992.

[62] I. Peter and G. Pietriek. Importance Driven Construction of Photon Maps. In *Rendering Techniques '98*, pages 269–280, 1998.

[63] M. Pharr, C. Kolb, R. Gershbein, and P. Hanrahan. Rendering Complex Scenes with Memory Coherent Ray Tracing. In *SIGGRAPH Conference Proceedings*, pages 101–108, 1997.

[64] R. Ramamoorthi and P. Hanrahan. Frequency Space Environment Map Rendering. In *Proceedings of SIGGRAPH 2002*, pages 517–526, 2002.

[65] M. Rudemo. Empirical Choice of Histograms and Kernel Density Estimators. In *Scand. J. Statist.*, volume 9, pages 65–78, 1982.

[66] S. R. Sain, K. A. Baggerly, and D. W. Scott. Cross-validation of Multivariate Densities. In *J. Am. Statist. Assoc.*, volume 89, pages 807–817, 1994.

[67] H. Samet. *Design and analysis of Spatial Data Structures: Quadtrees, Octrees, and other Hierarchical Methods.* Addison–Wesley, Redding, Mass., 1989.

[68] P. Schröder and W. Sweldens. Spherical Wavelets: Efficiently Representing Functions on the Sphere. In *Proceedings of SIGGRAPH '95*, pages 527–536, 1995.

[69] R. Schregle. Bias Compensation for Photon Maps. *Computer Graphics Forum*, 22(4):C792–C742, 2003.

[70] D. W. Scott and G. R. Terrell. Biased and Unbiased Cross-validation in Density Estimation. In *J. Amer. Statist. Assoc.*, volume 82, pages 1131–1146, 1987.

[71] P. Shirley, B. Wade, P. M. Hubbard, D. Zareski, B. Walter, and D. P. Greenberg. Global Illumination via Density Estimation. In *Rendering Techniques '95*, pages 219–230, 1995.

[72] P. Shirley, C. Wang, and K. Zimmerman. Monte Carlo Techniques for Direct Lighting Calculations. In *ACM Transactions on Graphics*, pages 1–36, 1996.

[73] F. X. Sillion, J. R. Arvo, S. H. Westin, and D. P. Greenberg. A Global Illumination Solution for general Reflectance Distribution. In *Proceedings of SIGGRAPH '91*, pages 187–196, 1991.

[74] B. Silverman. *Density Estimation for Statistics and Data Analysis.* Chapman and Hall, 1985.

[75] M. Smyk, S. Kinuwaki, R. Durikovic, and K. Myszkowski. Temporally Coherent Irradiance Caching for High Quality Animation Rendering. In *Proceedings of Eurographics*, volume 24, pages 401–412, 2005.

[76] F. Suykens. *On Robust Monte Carlo Algorithms for Multi-Pass Global Illumination.* PhD thesis, Department of Computer Science, Katholieke Universiteit Leuven, Leuven, Belgium, September 2002. Available from www.cs.kuleuven.ac.be/g̃raphics/CGRG.PUBLICATIONS/FRANKPHD/.

[77] F. Suykens and Y. D. Willems. Density Control for Photon Maps. In *Rendering Techniques 2000*, pages 23–34, 2000.

[78] E. Tabellion and A. Lamorlette. An Approximate Global Illumination System for Computer Generated Films. In *ACM Trans. Graph.*, volume 23, pages 469–476, 2004.

[79] D. A. Talbert and D. Fisher. An Empirical Analysis of Techniques for Constructing and Searching k-Dimensional Trees. In *Proceedings of the sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 26–33, 2000.

[80] T. Tawara, K. Myszkowski, K. Dmitriev, V. Havran, C. Damez, and H.-P. Seidel. Exploiting Temporal Coherence in Global Illumination. In *SCCG '04: Proceedings of the 20th Spring Conference on Computer Graphics*, pages 23–33, 2004.

[81] S. J. Teller. Computing the Antipenumbra of an Area Light Source. *Computer Graphics*, 26(2):139–148, July 1992.

[82] M. J. van Kreveld and M. H. Overmars. Divided kD-Trees. *Algorithmica*, 6(6):840–858, 1991.

[83] E. Veach. *Robust Monte Carlo Methods for Light Transport Simulation.* PhD thesis, Stanford University, December 1997. Available from http://graphics.stanford.edu/papers/veach_thesis/.

[84] E. Veach and L. J. Guibas. Metropolis Light Transport. In *Proceedings of SIGGRAPH '97*, pages 65–76, 1997.

[85] I. Wald, J. Günther, and P. Slusallek. Balancing Considered Harmful - Faster Photon Mapping Using the Voxel Heuristic. In *Computer Graphics Forum*, volume 22. Proceedings of Eurographics, 2004.

[86] I. Wald, T. Kollig, C. Benthin, A. Keller, and P. Slusallek. Interactive Global Illumination. Technical report, Computer Graphics Group, Saarland University, 2002.

[87] J. R. Wallace, M. F. Cohen, and D. P. Greenberg. A Two-Pass Solution to the Rendering Equation: A Synthesis of Ray Tracing and Radiosity Methods. In *Computer Graphics*, pages 311–320. Proceedings of SIGGRAPH '87, 1987.

[88] B. Walter, P. M. Hubbard, P. Shirley, and D. P. Greenberg. Global Illumination Using Local Linear Density Estimation. *ACM Transactions on Graphics*, 16(3):217–259, July 1997.

[89] M. Wand and M. Jones. *Kernel Smoothing.* Chapman and Hall, 1995.

[90] G. Ward. Adaptive Shadow Testing for Ray Tracing. In *Rendering Techniques '91*, Barcelona, Spain, 1991.

[91] G. Ward. Real Pixels. In J. Arvo, editor, *Graphic Gems II*, pages 80–83. Academic Press, 1991.

[92] G. J. Ward. Measuring and Modeling Anisotropic Reflection. In *Computer Graphics*, volume 26, pages 265–272. Proceedings of SIGGRAPH '92, 1992.

[93] G. J. Ward. The Radiance Lighting Simulation and Rendering System. In *Computer Graphics*, pages 459–472. Proceedings of SIGGRAPH '94, ACM Press, 1994.

[94] G. J. Ward and P. Heckbert. Irradiance Gradients. In *Rendering Techniques '92*, pages 85–98. Third Eurographics Workshop on Rendering, 1992.

[95] G. J. Ward, F. M. Rubinstein, and R. D. Clear. A Ray Tracing Solution for Diffuse Interreflection. In *Computer Graphics*, pages 85–92. Proceedings of SIGGRAPH '88, 1988.

[96] T. Whitted. An Improved Illumination Model for Shaded Display. In *Communication of the ACM*, volume 23, pages 343–349, 1980.

[97] A. Wilkie, R. F. Tobler, and W. Purgathofer. Orientation Lightmaps for Photon Radiosity in Complex Environments. In *Proceedings of CGI*, pages 318–327, 2000.

[98] J. Zaninetti, X. Serpaggi, and B. Péroche. A Vector Approach for Global Illumination in Ray Tracing. In *Computer Graphics Forum (Proceedings of Eurographics)*, volume 17(3), pages 149–158, 1998.