Pencil Light Transport

by

Mauro Steigleder

A thesis

presented to the University of Waterloo

in fulfilment of the

thesis requirement for the degree of

Doctor of Philosophy

in

Computer Science

Waterloo, Ontario, Canada, 2005

AUTHOR'S DECLARATION FOR ELECTRONIC SUBMISSION OF A THESIS

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Global illumination is an important area of computer graphics, having direct applications in architectural visualization, lighting design and entertainment. Indirect illumination effects such as soft shadows, color bleeding, caustics and glossy reflections provide essential visual information about the interaction of different regions of the environment. Global illumination is a research area that deals with these illumination effects. Interactivity is also a desirable feature for many computer graphics applications, especially with unrestricted manipulation of the environment and lighting conditions. However, the design of methods that can handle both unrestricted interactivity and global illumination effects on environments of reasonable complexity is still an open challenge.

We present a new formulation of the light transport equation, called *pencil light transport*, that makes progress towards this goal by exploiting graphics hardware rendering features. The proposed method performs the transport of radiance over a scene using sets of pencils. A pencil object consists of a center of projection and some associated directional data. We show that performing the radiance transport using pencils is suitable for implementation on current graphics hardware. The new algorithm exploits optimized operations available in the graphics hardware architecture, such as pinhole camera rendering of opaque triangles and texture mapping. We show how the light transport equation can be reformulated as a sequence of light transports between pencils and define a new light transport operator, called the *pencil light transport operator*, that is used to transfer radiance between sets of pencils.

Acknowledgements

Firstly, I would like to thank my advisor, Michael McCool, for all the insightful discussions and technical advice over all these years. I definitively learnt a lot while I was working with him.

I would also like to thank the Brazilian community of Waterloo for being always around. A special thanks goes to Carlson Cabral and Daniela Araújo for being my closest friends while I was in Waterloo.

Next I would like to thank my colleagues in the Waterloo Computer Graphics Lab. I own a great debt to Kevin Moule for helping me with several graphics hardware implementation problems. I must thank Stefanus Du Toit for being my Sh consultant during my last implementation port. I must also express my sincere appreciation to Tiberiu Popa for being a great friend and extraordinary colleague in the lab.

I would like to thank my parents, Geraldo and Laura Steigleder, for their love, support, and endless encouragement. Finally, a great deal of gratitude goes to my girlfriend, Patricia, for supporting me when I was *almost* giving up, and for making me believe I could make it.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Photorealistic image synthesis is an important area of computer graphics that has been in active development over a long period of time. The ability to generate images that have a similar appearance to the real world is one of the original motivating goals for computer graphics. Photorealism has important applications in areas like architectural visualization, lighting engineering, product design, visual effects, computer gaming and advertising. After several years of continuous improvements this research area has reached a reasonable maturity level with a sound foundation.

The correct simulation of illumination is a key component for the generation of photorealistic images. Indirect illumination effects such as color bleeding, soft shadows, caustics and glossy reflections provide essential visual information about the interaction of different regions of the environment. Global illumination is a research area that deals with these illumination effects, as well as many others. Global illumination algorithms

determine the overall steady-state light distribution of a scene by simulating the multiple interactions between light and objects of an environment.

Interactivity is also a desirable feature for many computer graphics applications, especially with unrestricted manipulation of arbitrary objects in the environment and lighting conditions. As faster hardware has become increasingly common, the demand for interactive applications including global illumination effects also increases. However, the design of methods that can handle both unrestricted interactivity and global illumination effects on environments of reasonable complexity is still an open challenge.

Classic global illumination algorithms for simulating global illumination, such as radiosity [1], path tracing [2], light tracing [3], bidirectional path tracing [4, 5] and photon mapping [6], have not been designed for interactive rendering. Some of these algorithms can achieve interactivity by implementing them on massively parallel architectures [7, 8], but they present scalability issues and require a costly investment on parallel machines. Several extensions to these algorithms have been developed in the last few years in order to achieve interactive frame rates. These extensions are mostly based on caching schemes, sampled over image [9, 10], object [11–13] and world [14–17] spaces, or based on incremental updates of partial global illumination solutions [18].

The main assumption of these caching extensions is that the light distribution is not subject to major changes between frames. Even though these techniques are able to achieve interactive rendering, their assumptions create a dependency on a large number of previously computed illumination samples. However, this assumption is broken by the

introduction of large changes in the lighting conditions or in the environment's geometry. Architectural design and modeling applications usually present such dynamic lighting and environment behaviours. Typical applications are normally interested in observing the results from generic changes in geometry and lighting conditions immediately after they are done. These changes can cause most of the precomputed illumination results to be discarded, requiring a large number of new illumination samples to be computed. The computation of new samples can considerably affect the overall time for the computation of the global illumination solution.

An alternative approach to the problem of achieving interactive rendering of environments including global illumination effects is to exploit high performance graphics hardware to improve the overall performance of light transport. The computational power of current Graphic Processing Units (GPU) is increasing at much faster rates than the computational power of Central Processing Units (CPU). Graphics hardware provides a highly parallel Single Instruction Multiple Data (SIMD) architecture composed of several specialized vector processors, each of them operating over a pipeline with a much larger number of levels than a standard CPU pipeline [19, 20]. Also, the Arithmetic and Logic Units (ALU) of a vector processor are able to execute complex floating-point operations over four parallel channels, including texture memory accesses with bilinear interpolation.

The lack of programmability used to be a major drawback for the use of graphics hardware in more complex applications. However, with the recent introduction of programmable vertex and fragment processors, graphics hardware has been used to solve

more general problems, like Fast Fourier Transforms [21], matrix multiplication [22, 23], systems of linear equations [24], fluid dynamics simulations [25, 26], and partial differential equations [27]. Even though current graphics hardware has higher programmability level, it still has several limitations, including difficulties with random memory writes and the relatively small number of instructions and registers per program. These deficiencies prevent the efficient management of the dynamic data structures that are usually required on many complex applications.

Nonetheless, several techniques have been developed that use the computational power of current GPUs to compute global illumination solutions. The first approaches to use graphics hardware for this purpose were generally limited to the number of illumination effects they could capture [28] or used a limited set of the available graphics hardware features [29, 30]. Recently, techniques have been developed to account for more complex global illumination effects, like caustics and specular reflections, exploiting more programmability features available on current graphics hardware. However, most of the techniques are restricted to simple GPU implementations of traditional global illumination algorithms, like ray tracing [31] and photon mapping [32, 33]. Also, most of these techniques still require a CPU-based preprocessing pass and are focused on interactive rendering of a given environment given a completed light simulation, instead of the transport of light over the environment or the acquisition of radiance field samples. Even though some techniques have been implemented completely on graphics hardware [34], the original algorithms are usually not designed to exploit the features that are optimized

on traditional GPUs, like rasterization, visibility test, and texture mapping.

The graphics hardware pipeline has been designed and intrinsically optimized to perform the standard pinhole camera rendering of triangles using rasterization and a depth buffer visibility test. The performance of these tasks have been continuously improved, with several optimization techniques having built-in support by many graphics card vendors. Techniques such as triangle strips, anti-aliasing, early occlusion culling, hierarchical Z-buffering, and lossless depth and color buffer compression, among others, have become commonly available on most current graphics hardware.

Also, since most techniques that exploit the GPU to simulate global illumination effects are based on traditional light transport algorithms, they are generally based on ray-tracing strategies. However, rendering using a rasterization approach has linear order of complexity with respect the number of polygons in the scene, and a sub-linear order with respect to the number of pixels on the screen. In contrast, rendering using ray-tracing approach has sub-linear order of complexity with respect to the number of polygons and linear with respect to the number of pixels in the screen. We would like to design an algorithm that can exploit the highly optimized rasterization and interpolation available hardware in graphics processing units.

Attribute interpolation is another important and widely used operation over the whole graphics hardware pipeline. Therefore interpolation units have become dedicated and highly optimized, with a large number of units available along the pipeline. Attribute interpolation is largely used during the rasterization stage and during texture lookup

operations.

This work presents a novel light transport formulation, called *pencil light transport*, which is suitable for direct implementation on current graphics hardware. The proposed method performs the transport of radiance over a scene using a set of points arbitrarily positioned in free space. The transport operator is based on the combination of two highly optimized graphics hardware operations: *standard pinhole camera rendering* (rasterization) and *texture reprojection*.

Each of these points distributed in free space are used to define an abstract primitive for light transport called *pencil object*. A pencil can be used to define a set of ray segments that pass through a common point, where each ray connecting two mutually visible surface points carries some information from one point to the other. By attaching radiance and other additional information to each of the rays specified by a pencil, it possible to simulate the light transport on a scene using pinhole camera rendering with local BRDF rendering and projective textures with shadows. Performing the light transport using a set of pencils enables the transfer of a large number of radiance samples over an environment simultaneously.

The main contribution of this thesis is to show how the light transport equation can be reformulated as a sequence of light transports between pencils. We define a new operator, called *pencil light transport operator*, that can be used to perform transfer of radiance between sets of pencils. We also show how this new operator can be efficiently mapped onto current graphics hardware.

The remainder of this work is organized as follows. Chapter 2 introduces some background information required for understanding the main issues involved in this research. Chapter 3 describes the most representative techniques developed for computing global illumination solutions at interactive rates as well as some global illumination algorithms that have been accelerated using graphics hardware.

Chapter 4 presents the pencil object as an abstraction for light transport and describes how its data structures as well as its functionality can be mapped to graphics hardware. Chapter 5 introduces the *pencil light transport* method and shows how the light transport equation can be reformulated using the pencil object abstraction as the base mechanism for light transport.

Chapter 6 describes details of a basic implementation of a pencil light transport algorithm and some refinements that can be used to improve the overall convergence of the algorithm. Finally, in chapter 7 we draw some conclusions and suggests possible extensions and areas of future research.

Chapter 2

Background

The light transport method described in this thesis uses concepts from different research areas. This chapter has the intention of describing briefly the most important principles from each of these areas, in enough detail to allow an understanding of the underlying ideas of pencil light transport. This chapter also defines the notation and nomenclature used in the remaining chapters.

There are several books that provide a more complete treatment of the research areas discussed on this background chapter. For the radiometry section, we suggest consulting the book *Thermal Radiation Heat Transfer* by Siegel et al. [35]. There are several books that cover light transport theory specifically in the context of computer graphics. We recommend the reading of Glassner's *Principle of Digital Image Synthesis* [36] for a detailed description. An alternative reference with more recent developments in the global illumination area is *Advanced Global Illumination* by Dutré et al. [37]. Monte Carlo

methods have been widely covered on several books. Classic references include the books from Hammersley and Handscomb [38] and Kalos and Whitlock [39]. Graphics hardware architectures and programming languages have been constantly changing over the last few years, so it is hard to recommend a standard reference. However, introductory material on GPU programming is covered *The OpenGL Programming Guide* by Shreiner [40]. Alternative sources for OpenGL programming also include *The OpenGL Graphics System: A Specification (Version 2.0)* by Segal and Akeley [41] and *The OpenGL Shading Language* by Kessenich et al. [42]. More advanced GPU programming techniques can be found in *GPU Gems* [43] and *GPU Gems 2* [44]. Our implementation was done using OpenGL and the Sh high-level GPU programming toolkit, so the book *Metaprogramming GPUs with Sh* by McCool and Du Toit [45] is also recommended.

2.1 Radiometry

Global illumination algorithms are mostly concerned with transporting light over an environment until a steady-state light distribution is reached. In order to understand the methods used to compute this light distribution as well as how they are related, it is important to understand the physical quantities used during light transport. This section describes the most common radiometric measures used in light transport techniques and how they relate to each other.

Radiometry is the area of study involved with the measurement of optical radiation, which accounts for electromagnetic radiation within the frequency range 3×10^{11} Hz

to 3×10^{16} Hz. Photometry is the area of study involved with measurement of the electromagnetic radiation as perceived by the human eye. Photometric measures are restricted to the wavelength range from about 360 to 830 nanometers [46]. Photometric units can be expressed by weighting the corresponding radiometric unit by the spectral response of the eye. A great deal of confusion concerns the misuse of some radiometric terms (such as intensity), so we will base the definitions used in this section on the definitions described by the *International System of Units* (SI) [47].

2.1.1 Geometric Definitions and Notations

Some radiometric units are defined with respect to geometric concepts, such as projected areas or solid angles. Before defining these radiometric units we therefore define the geometric measures they are based on.

In this thesis we denote points, either on surface or in free space, by boldface letters, like si: $\mathbf{x}$. We denote direction vectors by Greek letters, like so: ω. We differentiate between normalized and non-normalized vectors by using the symbols $\hat{\omega}$ and $\vec{\omega}$ for normalized and non-normalized vectors, respectively.

A *projected area* is defined as the area of the rectilinear projection of a surface of any shape onto the plane oriented in a given direction. For a differential area this amount

can be computed by:

$$dA^{\perp} = |\cos\theta|\, dA \tag{2.1}$$

$$= |\hat{\mathbf{n}}_a \cdot \hat{\mathbf{n}}|\, dA \tag{2.2}$$

where θ is the angle between the orientation of the differential area $\hat{\mathbf{n}}_a$ and the orientation of the plane of incidence $\hat{\mathbf{n}}$. For an arbitrary surface, the total projected area can be computed by integration,

$$A^{\perp} = \int_{\mathbf{x}\in A} |\cos\theta_{\mathbf{x}}|\, d\mathbf{x}\,, \tag{2.3}$$

where $\theta_{\mathbf{x}}$ is the angle between the orientation of the plane $\hat{\mathbf{n}}$ and the local surface normal $\hat{\mathbf{n}}_{\mathbf{x}}$ at surface point $\mathbf{x}$. Figure 2.1 shows the geometry of the projected area.

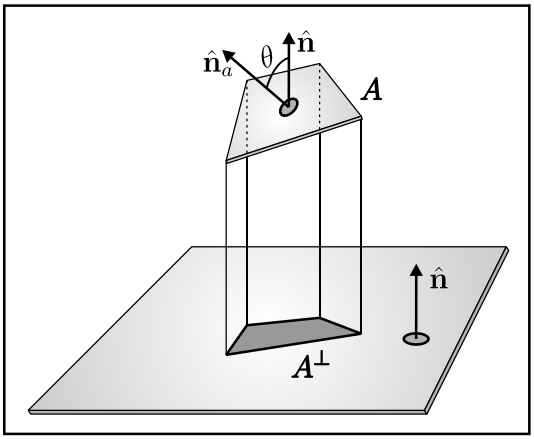


Figure 2.1: Geometry of the projected area. The differential area dA with local normal $\hat{\mathbf{n}}_a$, when projected on a plane oriented according to $\hat{\mathbf{n}}$, is given by $dA^{\perp}$.

The solid angle is another important geometric definition. Solid angles can be defined

similarly to the definition of a plane angle. The *National Institute of Standards and Technology* (NIST) defines one radian (rad) as the plane angle between two radii of a circle that cuts off on the circumference an arc equal in length to the radius [48]. That means a plane angle equals the length of the arc obtained by projecting a given curve to a unit circle.

The *solid angle* is an extension of the concept of plane angle to three dimensions. NIST also defines one steradian (sr) as the solid angle that, having its vertex in the center of a sphere, cuts off an area on the surface of the sphere equal to that of a square with sides of length equal to the radius of the sphere [48]. That means a solid angle equals the spherical area corresponding to the projection of a given object onto a unit sphere. Figure 2.2 shows the geometry of the solid angle.

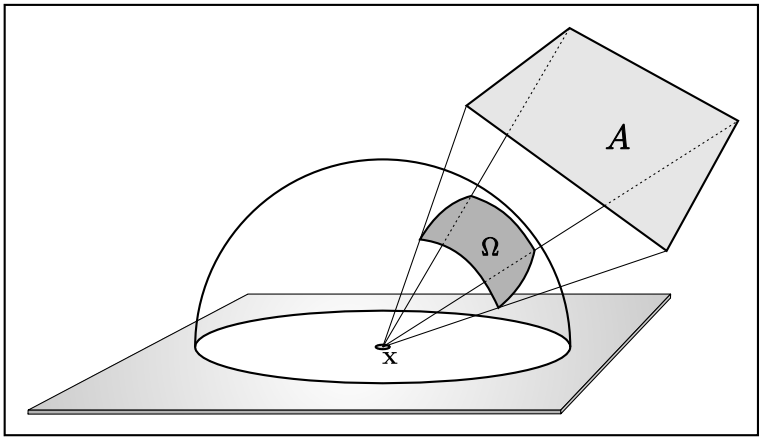


Figure 2.2: Geometry of the solid angle. The solid angle Ω is given by the area measure from projection of object A onto the unit sphere around $\mathbf{x}$.

A related definition commonly used by the computer graphics community is the *projected solid angle*. The projected solid angle is the area obtained by projecting the spheri-

cal surface defining the solid angle onto the base unit circle. The geometry of the projected solid angle is shown in figure 2.3.

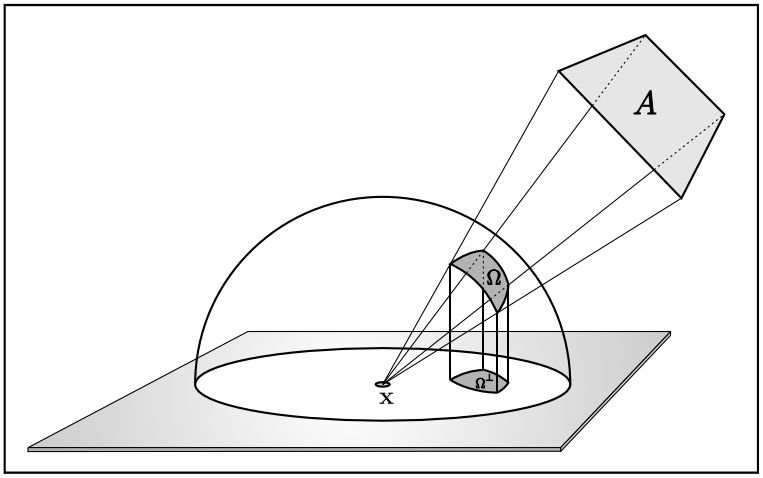


Figure 2.3: Geometry of the projected solid angle. The projected solid angle $\Omega^{\perp}$ is equivalent to projecting Ω onto the base plane at $\mathbf{x}$.

2.1.2 Radiometric Units

The two most elementary radiometric units are energy and power. *Energy* is an SI unit measured in Joules and often denoted by Q. Energy is defined as the capacity of a physical system to do work. *Power*, or *radiant flux*, is an SI unit that describes the rate of energy flow, that is, the amount of energy that arrives or leaves a surface per unit of time. The recommended symbol for power is Φ and it is measured in Watts (Joule/sec). Power can be expressed as the derivative

$$\Phi(t) = \frac{dQ(t)}{dt} \tag{2.4}$$

allowing us to express energy as the integral of power over time:

$$Q(t) = \int \Phi(t)\, dt \tag{2.5}$$

When describing the interaction between energy and surfaces, it is useful to have measures that include geometric quantities, like area and solid angle. *Irradiance*, also known as *flux density*, is defined as the power per unit area, incident from all directions of a hemisphere, onto an oriented surface that coincides with the base of that hemisphere. *Radiant exitance*, or *radiosity*, is a related measure defined as the total power leaving a surface into a hemisphere whose base is on that surface. The symbols used for irradiance and radiosity are E and B, respectively. Both measures can be expressed as the derivative of power with respect to a projected surface area:

$$E = \frac{d\Phi_i}{dA^{\perp}} \qquad \text{and} \qquad B = \frac{d\Phi_o}{dA^{\perp}} \tag{2.6}$$

where we distinguish the measures using Φ_i and Φ_o to represent incident and exitant power, respectively. Both quantities are measured in Watt/m^2. Integrating the irradiance or the radiant exitance over the corresponding area results in the power arriving or leaving a surface.

Irradiance and radiant exitance are radiometric units defined at a surface area. In order to describe the radiometric properties of point light sources, a different radiometric unit is desirable. *Radiant intensity*, represented by the symbol I, describes the power

leaving a point per unit solid angle:

$$I = \frac{d\Phi}{d\omega} \tag{2.7}$$

and is measured in Watt/sr. Similarly to the irradiance and radiant exitance, integrating the radiant intensity over the total solid angle around a point results in the total power leaving that point.

Radiance

Radiance is a fundamental radiometric measure and describes the radiant power arriving or leaving a surface point per unit solid angle, per unit projected area. Radiance is measured in Watts/m^{2}sr and is represented by the symbol L. It can be expressed as

$$L = \frac{d^2\Phi}{d\hat{\omega}\ dA^{\perp}} = \frac{d^2\Phi}{d\hat{\omega}\ dA\ \cos\theta} = \frac{d^2\Phi}{d\hat{\omega}^{\perp}\ dA} \tag{2.8}$$

where $dA^{\perp}$ is the differential projected area relative to the direction $\hat{\omega}$, and θ is the angle between $\hat{\omega}$ and the local surface area at dA. Similarly, $\hat{\omega}^{\perp}$ is the differential projected solid angle with respect to the plane defined by dA. Figure 2.4 shows the geometry of a radiance measurement.

Radiance is usually the unit of choice in light transport algorithms because it has a close relationship to the brightness of individual points in a scene as perceived by the human eye. Also, radiance has some properties that make it suitable for simulation of light transfer over a scene.

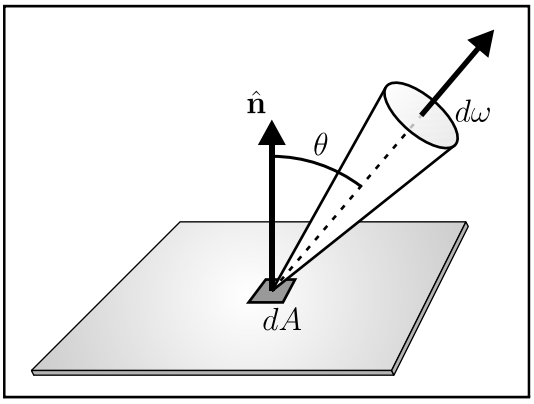


Figure 2.4: Geometry of radiance. Radiance defines the power flux leaving the differential surface area dA in the direction given by the differential solid angle $d\hat{\omega}$.

One of these properties is that radiance is constant along a ray travelling in a vacuum (or in clear air over human scales). That means all points along a straight line joining two mutually visible surface points have the same radiance measure. Figure 2.5 illustrates the straight line invariance property. The radiance leaving the point $\mathbf{x}$ in direction $\hat{\omega}$ is the same for all points $\mathbf{x}_i$ positioned in the line joining $\mathbf{x}$ and $\mathbf{y}$, for the same direction $\hat{\omega}$.

If we define a function over all possible distinct radiance measures of a scene, this function would have four degrees of freedom: two degrees for defining the orientation of incidence or exitance, and two degrees for defining the position on a surface (assuming that all surfaces in the scene can be parameterized onto a plane). Even though this parameterization is minimal, sometimes redundancy is preferred for generality. It is possible to define a simpler version of the previously defined function so that no surface parameterization is necessary. By allowing radiance to be specified over all points in a

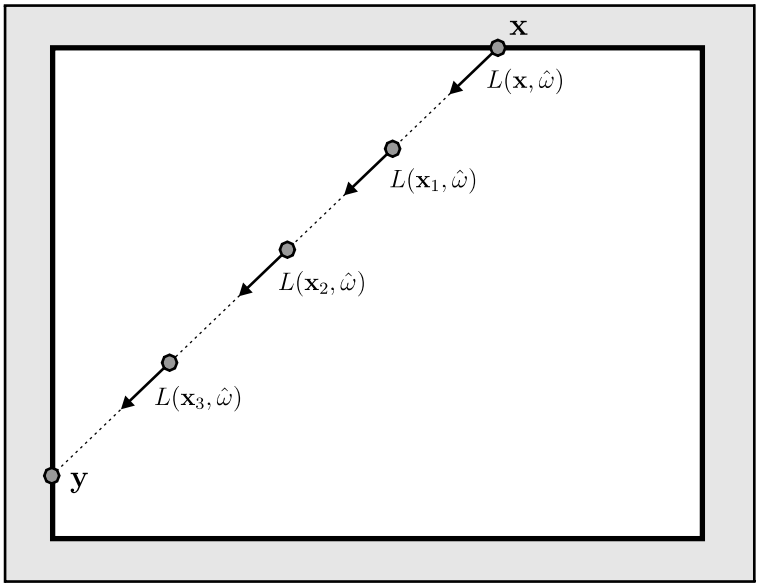
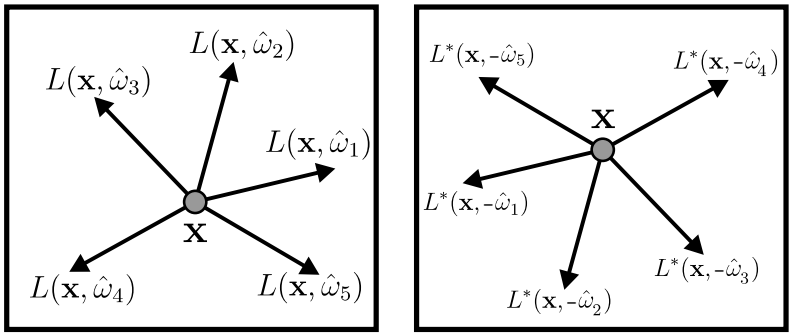


Figure 2.5: The radiance leaving points $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$ in the direction $\hat{\omega}$ is the same as the radiance leaving $\mathbf{x}$ in the same direction.

scene (including points in free space), a function $L(\mathbf{x}, \hat{\omega})$ with five degrees of freedom can be defined. We refer to this function as the *exitant radiance field* [37]. A function $L(\mathbf{x}, \hat{\omega})$ specifies the radiance leaving point $\mathbf{x}$ in direction $\hat{\omega}$. Conversely, we define a function $L^*(\mathbf{x}, \hat{\omega})$ as the *incident radiance field* [37], which specifies the radiance arriving at a given point $\mathbf{x}$ from a direction $\hat{\omega}$. Figure 2.6 exemplifies the notation used for exitant and incident radiance fields.

Only one radiance field is strictly necessary, since either $L(\mathbf{x}, \hat{\omega})$ or $L^*(\mathbf{x}, \hat{\omega})$ can be obtained from each other. However, this separation will be useful when expressing light transport equations. An important property relating these radiance fields is that, in the



(a) (b)

Figure 2.6: (a) Samples of the *exitant* radiance field $L(\mathbf{x}, \hat{\omega})$ at a fixed point $\mathbf{x}$. (b) Samples from the *incident* radiance field $L^*(\mathbf{x}, \hat{\omega})$ at a fixed point $\mathbf{x}$.

absence of a participating medium, they are related by the equality

$$L(\mathbf{x}, \hat{\omega}) = L^*(\mathbf{x}, -\hat{\omega}) \tag{2.9}$$

for all points $\mathbf{x}$ in free space. Equation 2.9 states that for an arbitrary point located in free space, the incident radiance from a given direction is equivalent to the exitant radiance toward the opposite direction. This equality comes directly from the property that radiance is invariant along straight lines, in the absence of participating medium. A medium is called participating when the amounts of absorption, scattering or dispersion of light along rays is not negligible [36].

2.1.3 Bidirectional Reflectance Distribution Function

When light is transported over a scene, it eventually strikes surfaces and is reflected, transmitted or absorbed. In order to model this interaction between light and surfaces,

a special function must be defined. In its general form, this function is called the *Bidirectional Scattering-Surface Reflectance Distribution Function* (BSSRDF) and describes the amount of radiance arriving from a given incoming direction at a surface point that is reflected towards an outgoing direction from a outgoing point [49]. The *Bidirectional Reflectance Distribution Function* (BRDF) provides a simplified representation of the BSSRDF, ignoring subsurface scattering and assuming invariance on the point of observation.

Since radiance is defined with respect to a given point and direction, it is sometimes common to define a BRDF as a function of position as well. This BRDF is commonly called *spatial BRDF* and can be specified using a combination of BRDFs [45]. Formally, the spatial BRDF can be defined, with respect to a surface point $\mathbf{x}$, as the differential ratio of radiance leaving in an outgoing direction and the radiance density arriving from a differential incoming solid angle:

$$f_r(\hat{\omega}_i, \mathbf{x}, \hat{\omega}_o) = \frac{dL(\mathbf{x}, \hat{\omega}_o)}{L^*(\mathbf{x}, \hat{\omega}_i) \cos\theta_i \; d\omega_i} \tag{2.10}$$

where the outgoing and incident directions are given by $\hat{\omega}_o$ and $\hat{\omega}_i$, respectively, and θ_i is the angle between the local surface normal at $\mathbf{x}$ and $\hat{\omega}_i$. Figure 2.7 illustrates the geometry of a BRDF.

There are some properties about the BRDF that are important to mention. First, since the denominator in the BRDF definition is a radiance density, this allows for possible

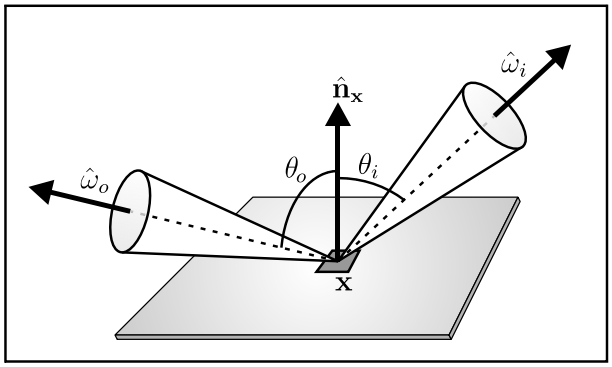


Figure 2.7: A BRDF defines the amount of the radiance density coming from direction $\hat{\omega}_i$ that is reflected in the direction $\hat{\omega}_o$.

values of the BRDF in the range from 0 to infinity.

Reciprocity is a desirable property for a BRDF because it allows for more flexible implementations of light transport methods. A BRDF is called reciprocal when it respects the *Helmholtz condition*. The Helmholtz condition states that inverting the direction of radiance propagation has no influence on the amount and distribution of radiance that is transported, or in other words,

$$f_r(\omega_i, \mathbf{x}, \omega_o) = f_r(\omega_o, \mathbf{x}, \omega_i)\,. \tag{2.11}$$

Even though many BRDF used in computer graphics applications are reciprocal, there are real materials that do not satisty the Helmholtz condition [50].

Another desirable property for a BRDF related to implementation of light transport methods is *plausability* [51]. For a BRDF to be *plausible*, in addition to satisfying reciprocity, it must also obey the energy conservation property. The energy conservation law

says the total amount of power reflected at any given point must be less or equal to the total amount of incident power. This property can be formally expressed as:

$$\forall \omega_i : \int_{\Omega_{\mathbf{x}}} f_r(\omega_i, \mathbf{x}, \omega_o) \cos\theta_o \, d\omega_o < 1 \tag{2.12}$$

where $\Omega_{\mathbf{x}}$ is the set of directions on the forward hemisphere at point $\mathbf{x}$ and θ_o is the angle between the local surface normal at point $\mathbf{x}$ and the outgoing direction $\hat{\omega}_o$. The product of the BRDF by its associated cosine term is a common expression in global illumination algorithms and is usually called the *Scattering Probability Function* (SPF) [52]:

$$f_r(\omega_i, \mathbf{x}, \omega_o) = C \; s_r(\omega_i, \mathbf{x}, \omega_o) \cos\theta_o \tag{2.13}$$

where the constant C ensures the condition that the SPF integrates to unity. The SPF specifies the fraction of the incident radiance that is scattered towards a specific outgoing diretion.

2.2 The Light Transport Problem

The main goal of a light transport algorithm is to compute a set of measurements corresponding to the steady-state distribution of light over an environment. These measurements account for the radiance arriving at a set of sensors from a given direction, after repeated interaction with surfaces in this environment. The light transport formulation

used in this thesis is simplified, leaving out illumination effects like participating media, subsurface scattering, polarization, phosphorescence and fluorescence [36].

One of the most common formulations of the light transport problem is the *hemispherical formulation*. Also referred as the *rendering equation* in the computer graphics community [2], it is based on the definition of the BRDF and can be derived as follows. The exitant radiance at a given point $\mathbf{x}$ toward a given direction $\hat{\omega}_o$ can be expressed as the emitted radiance plus the scattered radiance at $\mathbf{x}$ in direction $\hat{\omega}$:

$$L(\mathbf{x}, \hat{\omega}_o) = L_e(\mathbf{x}, \hat{\omega}_o) + L_r(\mathbf{x}, \hat{\omega}_o) \tag{2.14}$$

The scattered radiance $L_r(\mathbf{x}, \hat{\omega}_o)$ can be computed based on equation 2.10 as

$$L_r(\mathbf{x}, \hat{\omega}_o) = \int_{\Omega_{\mathbf{x}}} f_r(\hat{\omega}_i, \mathbf{x}, \hat{\omega}_o) L^*(\mathbf{x}, \hat{\omega}_i) \cos\theta_i \, d\omega_i \tag{2.15}$$

However, it is desirable to operate using the same radiance field. Considering that radiance is constant along straight lines, the incoming radiance $L^*(\mathbf{x}, \hat{\omega}_i)$ can be expressed using the exitant radiance by

$$L^*(\mathbf{x}, \hat{\omega}_i) = L(h(\mathbf{x}, \hat{\omega}_i), -\hat{\omega}_i) \tag{2.16}$$

where $h(\mathbf{x}, \hat{\omega})$ is the ray casting function and returns the surface closest point to $\mathbf{x}$ in the direction $\hat{\omega}$. Figure 2.8 shows the geometry of equation 2.16.

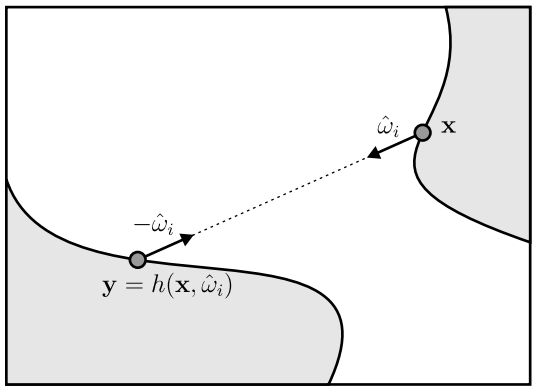


Figure 2.8: The incident radiance at $\mathbf{x}$ arriving from direction $-\hat{\omega}_i$ is equivalent to the exitant radiance at $\mathbf{y} = h(\mathbf{x}, \hat{\omega}_i)$ in the same direction.

Putting equations (2.14), (2.15) and (2.16) together yields the final hemispherical formulation of the rendering equation:

$$L(\mathbf{x}, \hat{\omega}_o) = L_e(\mathbf{x}, \hat{\omega}_o) + \int_{\Omega_\mathbf{x}} f_r(\hat{\omega}_i, \mathbf{x}, \hat{\omega}_o) L(h(\mathbf{x}, \hat{\omega}_i), -\hat{\omega}_i) \cos\theta_i \, d\omega_i \tag{2.17}$$

Figure 2.9 shows the geometry of the hemispherical formulation of the light transport problem equation. An alternative formulation to the hemispherical formulation of the light transport problem is the *area formulation*. Instead of performing the integration over the hemisphere around a surface point, the area formulation of light transport accounts for the contribution of all possible surface points in the scene, performing the integration over all differential surface areas in the environment [52].

The area formulation can be constructed based on the hemispherical formulation defined in equation 2.17. In order to make the conversion to the new domain of integration, consider the following equivalence between a differential area $dA_\mathbf{y}$ and its corresponding

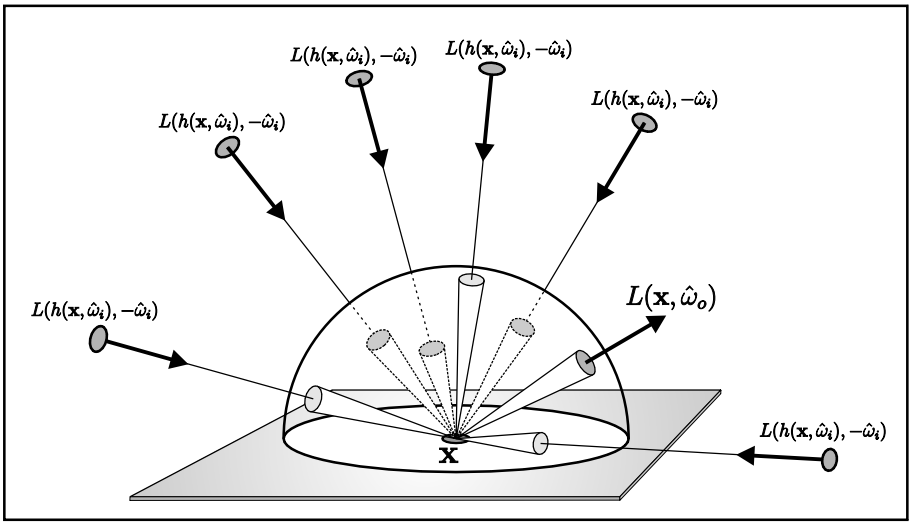


Figure 2.9: Hemispherical formulation of the light transport problem using exitant radiance.

differential solid angle at a point $\mathbf{x}$:

$$d\hat{\omega} = \frac{dA^{\perp}_{\mathbf{y}}}{r^2} = \frac{|\cos\theta_{\mathbf{y}}|\, dA_{\mathbf{y}}}{r^2} \tag{2.18}$$

where $\theta_{\mathbf{y}}$ is the angle between the surface normal at point $\mathbf{y}$ and the direction from point $\mathbf{y}$ to point $\mathbf{x}$. The distance between surface points $\mathbf{x}$ and $\mathbf{y}$ is given by r.

Combining equations (2.18) and (2.17) results in the following area formulation of light transport:

$$L(\mathbf{x},\hat{\omega}_o) = L_e(\mathbf{x},\hat{\omega}_o) + \int_{\mathcal{S}} f_r(\overrightarrow{\mathbf{xy}},\mathbf{x},\hat{\omega}_o) L(\mathbf{y},\overrightarrow{\mathbf{yx}}) V(\mathbf{x},\mathbf{y}) \frac{\cos_+\theta_{\mathbf{x}}\cos_+\theta_{\mathbf{y}}}{r^2}\, dA_{\mathbf{y}} \tag{2.19}$$

where $\mathcal{S}$ is the set of all surface points in the scene, and $\theta_{\mathbf{x}}$ and $\theta_{\mathbf{y}}$ are the angles between

the surface normals at points $\mathbf{x}$ and $\mathbf{y}$. The direction of radiance flux from a differential surface $dA_{\mathbf{y}}$ is defined by $\overrightarrow{\mathbf{xy}}$. The function $V(\mathbf{x}, \mathbf{y})$ is the *visibility function* and returns 1 if the points $\mathbf{x}$ and $\mathbf{y}$ are mutually visible and 0 otherwise. The function

$$\cos_+ \theta = \max(\cos\theta, 0) \tag{2.20}$$

is introduced in order to guarantee that radiance is only transferred between forward hemispheres of differential areas. Figure 2.10 shows the geometry of the area formulation of the rendering equation.

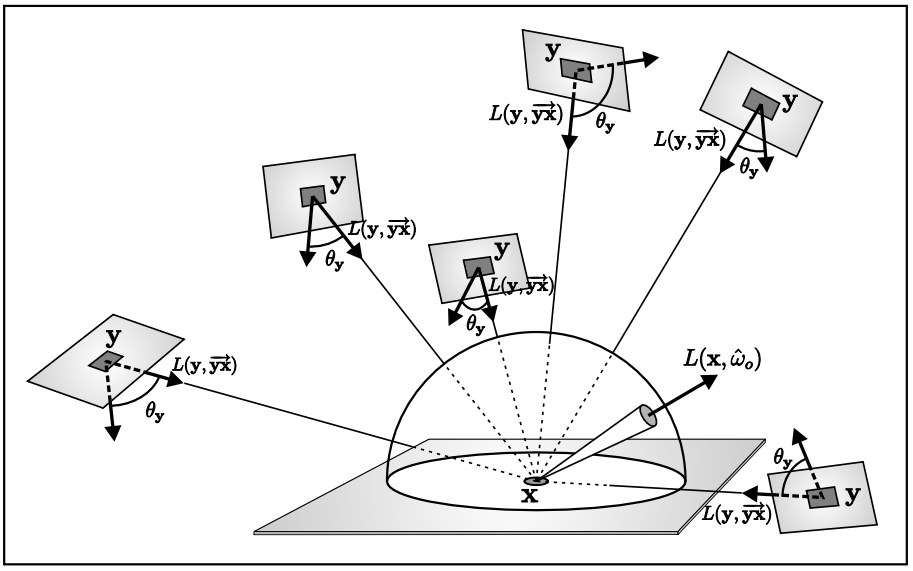


Figure 2.10: Area formulation of the light transport problem using exitant radiance field.

It is also possible to express the light transport problem with respect to the incident radiance field. In this case the hemispherical formulation of the light transport problem

becomes

$$L^*(\mathbf{x}, \hat{\omega}_x) = L_e^*(\mathbf{x}, \hat{\omega}_x) + \int_{\Omega_\mathbf{y}} f_r(\hat{\omega}_y, \mathbf{y}, -\hat{\omega}_x) L^*(\mathbf{y}, -\hat{\omega}_y) \cos\theta_\mathbf{y} \, d\omega_y, \tag{2.21}$$

where, for simplicity, the point $\mathbf{y}$ is defined as the closest surface point in the direction $\hat{\omega}_x$ and can be computed by $\mathbf{y} = h(\mathbf{x}, \hat{\omega}_x)$, the set of directions on the hemisphere around $\mathbf{y}$ is specified by $\Omega_\mathbf{y}$, and $\theta_\mathbf{y}$ is the angle between the surface normal at $\mathbf{y}$ and the line joining $\mathbf{y}$ and $\mathbf{x}$. Figure 2.11 shows the geometry of the hemispherical formulation of the light transport using incident radiance field.

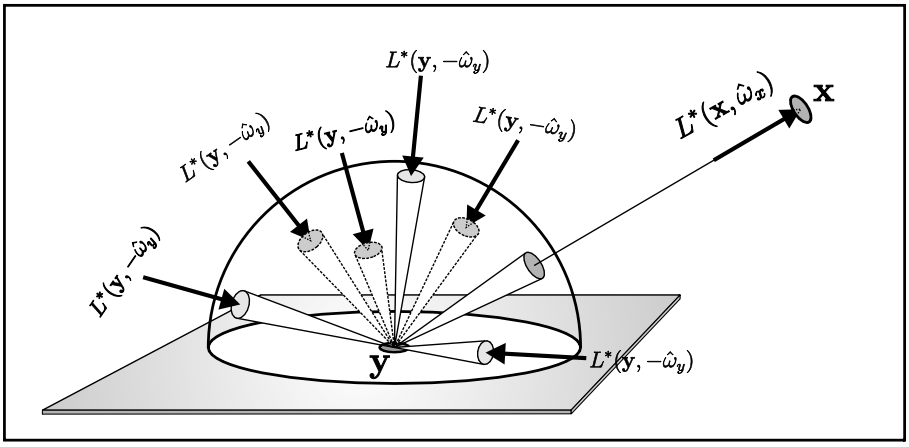


Figure 2.11: Hemispherical formulation of the light transport problem using incident radiance

Hemispherical formulation of the light transport problem using incident radiance.

The light transport problem can also be expressed using the area formulation over

the incident radiance field, resulting in the following equation

$$L^*(\mathbf{x}, \hat{\omega}_x) = L_e^*(\mathbf{x}, \hat{\omega}_x) + \int_{\mathcal{S}} f_r(\overrightarrow{\mathbf{yz}}, \mathbf{y}, \hat{\omega}_x) L^*(\mathbf{y}, \overrightarrow{\mathbf{zy}}) V(\mathbf{y}, \mathbf{z}) \frac{\cos_+ \theta_\mathbf{y} \cos_+ \theta_\mathbf{z}}{r^2} \, dA_\mathbf{z}. \quad (2.22)$$

where $\hat{\omega}_x$ is the incident direction at $\mathbf{x}$, and the point $\mathbf{y} = h(\mathbf{x}, \hat{\omega}_x)$ is the closest surface point from $\mathbf{x}$ in the direction $\hat{\omega}_x$. The direction of incident radiance coming from a differential surface $dA_\mathbf{z}$ to the surface point $\mathbf{y}$ is given by $\overrightarrow{\mathbf{zy}}$. The angles between the line joining $\mathbf{y}$ and $\mathbf{z}$ and the surface normals at these endpoints are given by $\theta_\mathbf{y}$ and $\theta_\mathbf{z}$, respectively. The distance between surface points $\mathbf{y}$ and $\mathbf{z}$ is given by r. Figure 2.12 shows the geometry of the area formulation of light transport using incident radiance field.

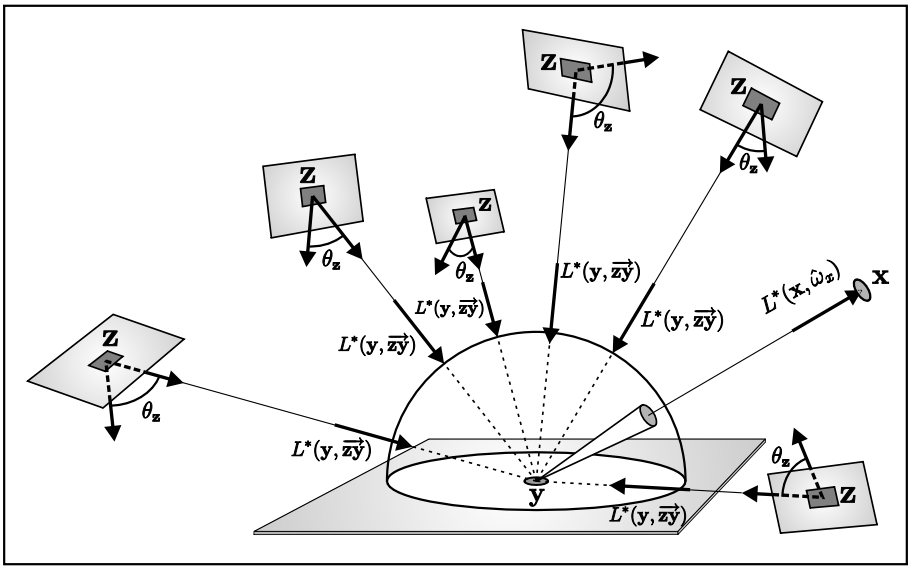


Figure 2.12: Area formulation of the light transport problem using incident radiance.

The difference between light transport formulations using the exitant and the incident

radiance fields can be understood intuitively. When operating over the exitant radiance field, the aim is to determine the total amount of radiance that comes from the environment and is reflected towards a particular direction after hitting a surface point. When operating over the incident radiance field, the aim is to determine the amount of radiance that is reflected by the whole environment and arrives at a given surface point from a particular direction.

Equations (2.17) and (2.19) define the light transport problem as an implicit integral equation using the exitant radiance field. More specifically, as a Fredholm integral equation of the second kind, since the unknown quantity $L(\mathbf{x}, \hat{\omega})$ is located on both the left-hand and right-hand sides of the equation, being part of the integrand on the right-hand side [36]. This Fredholm equation is of the second kind because the right-hand side also contains an additional term independent of the unknown quantity [36]. The objective of a global illumination algorithm is to solve these equations to find the unique function $L(\mathbf{x}, \hat{\omega})$, the steady-state radiance field.

The hemisphere and area formulations of the light transport problem provide a good description of the light transport process. However, a more concise notation is useful when we analyze the properties of a global illumination solution, or if we want to develop a new formulation. Fortunately, these equations can be expressed in a more concise form using a functional notation.

Let the set of all functions with domain X be defined by $G(X)$. In functional analysis, an operator $\mathcal{A} : G(X) \rightarrow G(X)$ is defined as a mapping that assigns to every function

$g(X) \in G(X)$ a function $A(g(X)) \in G(X)$ [53]. Informally, an operator can be understood as a special function that operates on functions, returning another function as a result. The notation $k = \mathcal{A}\ g$ denotes the application of an operator $\mathcal{A}$ to a function g resulting in another function k. An operator is called a *linear operator* if it satisfies the conditions $\mathcal{A}(g + k) = \mathcal{A}\ g + \mathcal{A}\ k$ and $\mathcal{A}(\alpha g) = \alpha(\mathcal{A}\ g)$ [53].

The radiance field L can be treated as an element of a function space $G(X)$, where the domain X is defined by the set of all possible points and directions in the scene. In this case, the scattering integral in the rendering equation can be expressed as a linear operator over the radiance field. This operator is called the *light transport operator* and is denoted by $\mathcal{T}$. For the hemispherical formulation, it can be defined by

$$(\mathcal{T}g)(\mathbf{x}, \hat{\omega}_o) = \int_{\Omega_{\mathbf{x}}} f_r(\hat{\omega}_i, \mathbf{x}, \hat{\omega}_o)\ g(h(\mathbf{x}, \hat{\omega}_i), \hat{\omega}_i) \cos\theta_i\ d\hat{\omega}_i \tag{2.23}$$

The light transport operator is a linear operator that when applied to the radiance field L returns the radiance field distribution after one light bounce. It is important to observe that the light transport operator $\mathcal{T}$ is applied to the entire radiance field L, not only to a single radiance field entry $L(\mathbf{x}, \hat{\omega})$. Using the light transport operator defined in equation 2.23, the light transport problem can be expressed as

$$L = L_e + \mathcal{T}L. \tag{2.24}$$

The goal of a light transport algorithm is to solve equation 2.24 for the radiance field

L (or for a subdomain of L). Using the functional notation, this solution can be expressed symbolically by isolating the term L in 2.24:

$$L = (\mathcal{I} - \mathcal{T})^{-1} L_e \tag{2.25}$$

where $\mathcal{I}$ is the *identity operator*.

The functional form allows to symbolically manipulate the light transport problem in order to arrive at different methods to describe a global illumination solution [50, 54]. Equation 2.24 can be recursively substituted on itself, allowing us to express the global illumination solution as an infinite Neumann series:

$$L = L_e + \mathcal{T}L_e + \mathcal{T}^2 L_e + \mathcal{T}^3 L_e + \cdots = \sum_{i=0}^{\infty} \mathcal{T}^i L_e \tag{2.26}$$

where $\mathcal{T}^i$ represent the repeated application of the light transport operator for i times. Using equations 2.25 and 2.26 it is possible to define the following relation for the light transport operator

$$\sum_{i=0}^{\infty} \mathcal{T}^i = (\mathcal{I} - \mathcal{T})^{-1} \tag{2.27}$$

Both equations 2.25 and 2.26 only converge if $|\mathcal{T}| < 1$, where $|\mathcal{T}|$ is the norm of the light transport operator. The norm $|\mathcal{T}|$ has a physical meaning, representing the average reflectance of surfaces in a scene. This condition is true as long as the BRDFs used to describe the reflectance properties of the scene are plausible. Note the formal similarity

of relation 2.27 and the solution to the geometric series

$$\sum_{i=0}^{\infty} r^i = \frac{1}{1-r} \tag{2.28}$$

which converges only if $|r| < 1$.

Each term in the Neumann series has also a physical interpretation: $\mathcal{T}^i L_e$ is the radiance field distribution after i reflections of the emitted radiance. In other words, $\mathcal{T}^0 L_e = L_e$ is the self-emitted radiance field, $\mathcal{T}^1 L_e = \mathcal{T} L_e$ is the radiance field after one light bounce, $\mathcal{T}^2 L_e$ is the radiance after two light bounces, and so on. Adding all individual radiance field distributions $\mathcal{T}^i L_e$ results in the global illumination solution.

It is possible to define the multiple-bounce light transport operator recursively as follows:

$$\mathcal{T}^n L_e = \mathcal{T}\left(\mathcal{T}^{n-1} L_e\right) \tag{2.29}$$

$$\mathcal{T}^0 L_e = L_e \tag{2.30}$$

A particularly useful application of the functional notation is the reformulation of the light transport operator proposed by Veach [50]. Veach [50] has shown that the light transport operator $\mathcal{T}$ can be conviniently expressed as an alternation of a scattering operator $\mathcal{K}$ and a propagation operator $\mathcal{G}$.

The scattering operator $\mathcal{K}$ describes the interaction between light and surfaces, defin-

ing the transformation between the total incoming radiance and the radiance that is reflected into a specific outgoing direction. This scattering operator is defined as

$$(\mathcal{K}g^*)(\mathbf{x},\hat{\omega}_o) = \int_{\Omega_\mathbf{x}} f_r(\hat{\omega}_i,\mathbf{x},\hat{\omega}_o)\; g(\mathbf{x},\hat{\omega}_i)\cos\theta_i\; d\hat{\omega}_i \tag{2.31}$$

where $g^*(\mathbf{x},\hat{\omega})$ is a function representing the *incoming* radiance field arriving from direction $\hat{\omega}$ at point $\mathbf{x}$, and $g(\mathbf{x},\hat{\omega})$ is a function representing the *exitant* radiance field leaving in direction $\hat{\omega}$ at point $\mathbf{x}$.

We note that both fields g and g^* are defined over the same domain. We use different notations for each one only to distinguish between their semantics. Function g^* is used to specify a field that describes an incident radiance, while g is used to specify a field that descibes an exitant radiance. That means operator $\mathcal{K} : L^* \rightarrow L$ in equation 2.31 is executed over the incident radiance field $L^*(\mathbf{x},\hat{\omega})$ and yields as result a function that describes the exitant radiance field $L(\mathbf{x},\hat{\omega})$ for the same domain.

One important property of the scattering operator is that it is defined *locally* at every point $\mathbf{x}$, so the computational cost for evaluating the scattering operator is independent of the scene's geometric complexity and organization. The scattering operator defines the relationship between incoming and exitant radiance fields at fixed points $\mathbf{x}$. Similarly, the propagation operation describes the radiance flow between two surface points and provides a relationship between incoming and exitant radiance fields for fixed directions

$\hat{\omega}$. The propagation operator $\mathcal{G}$ is defined as

$$(\mathcal{G}g)(\mathbf{x},\hat{\omega}) = \begin{cases} g^*(h(\mathbf{x},\hat{\omega}),-\hat{\omega}) & \text{if } d(\mathbf{x},\hat{\omega}) < \infty \\ 0 & \text{otherwise} \end{cases} \tag{2.32}$$

where $d(\mathbf{x},\hat{\omega})$ is the *boundary distance function* and returns the distance from $\mathbf{x}$ to the closest surface in the direction $\hat{\omega}$.

In contrast to the scattering operator $\mathcal{K}$, the propagation operator $\mathcal{G}$ is not defined locally. This implies a dependency between the computational cost for its evaluation and the scene complexity and arrangement. Figure 2.13 illustrates the radiance transport resulting from the application of a sequence of propagation and scattering operators. Also, operator $\mathcal{G} : L \rightarrow L^*$ executes over an exitant radiance field and produces as result a field that has the semantic of an incident radiance field.

Based on the definitions of the scattering and propagation operators, the light transport operator can then be defined using operator composition

$$\mathcal{T} = \mathcal{K} \circ \mathcal{G}. \tag{2.33}$$

The separability of the light transport operator into a scattering operator and a propagation operator provides a useful abstraction. Since the global propagation operator has higher complexity order than the local scattering operator, it is useful to be able to handle each operator independently. The propagation operator is a key component of the light

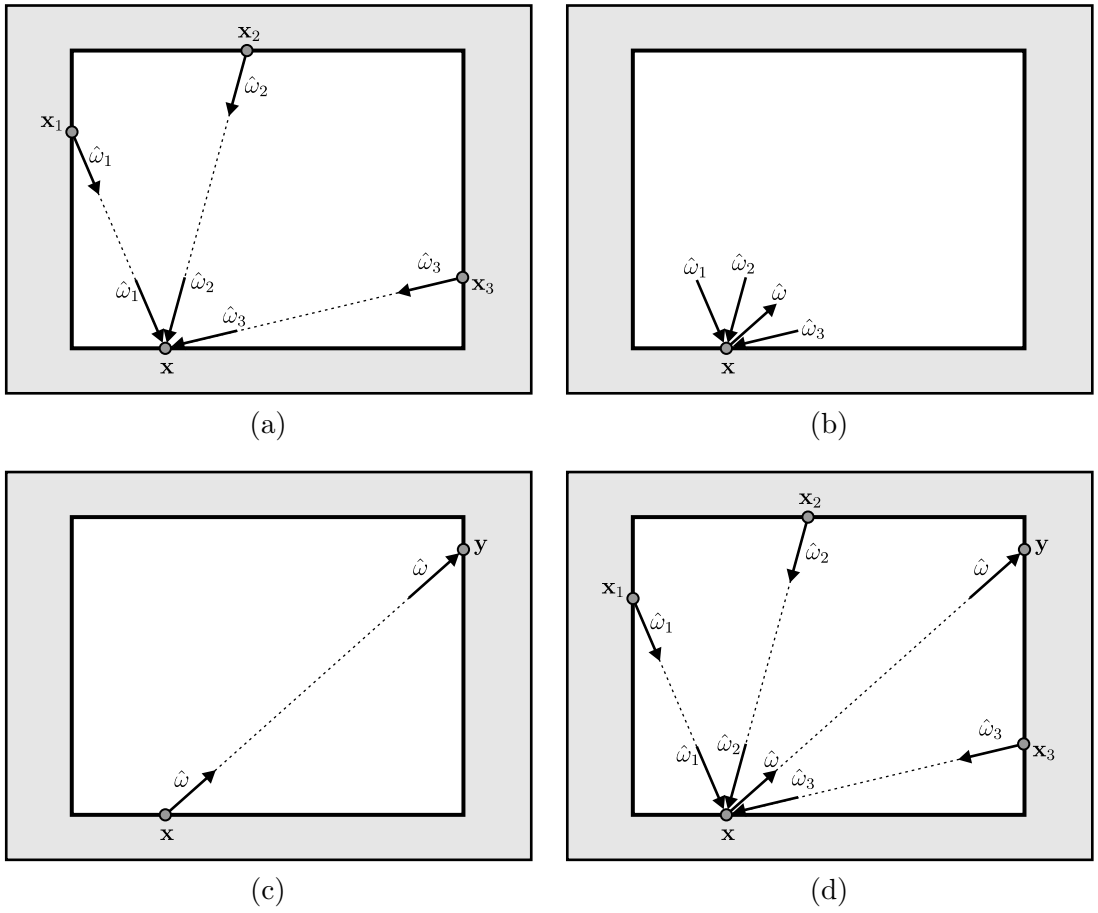


Figure 2.13: Propagation and scattering operators. (a) Propagation operator applied to exitant radiance samples. (b) Scattering operator applied to incident radiance samples $L^*(\mathbf{x}, \hat{\omega}_1)$, $L^*(\mathbf{x}, \hat{\omega}_2)$ and $L^*(\mathbf{x}, \hat{\omega}_3)$. (c) Propagation operator applied to radiance sample $L(\mathbf{x}, \hat{\omega})$. (d) Total radiance transport given by the application of $\mathcal{G} \circ \mathcal{K} \circ \mathcal{G}$ corresponding to (a), (b) and (c).

transport algorithm and eventually determines the overall performance and scalability of the algorithm. Handling each operator individually allows for the development of efficient methods to implement the propagation operator, without having to consider specific details of the scattering operator implementation [55]. Also, in a real rendering system, the scattering operator may include several different reflectance models, specialized shaders, texture mapping and many other local material properties. In this scenario, it becomes useful to isolate the scattering operator and define a clear interface with the propagation operator.

2.3 Numerical Integration Methods

There are several deterministic quadrature methods for numerical integration described in the standard literature that can achieve low error as well as high convergence rates. However, these methods are based on the strong assumptions that the domain of integration has low dimensionality and that the integrand is continuous and has continuous derivatives. From the descriptions given in the previous section, we can observe that the multiple-light bounce light operator $\mathcal{T}^i L_e$ describes a multiple integral over i hemispheres. That makes the domain of integration unusably high for numerical quadrature methods, even for a relatively small number of light bounces. Also, due to occlusion, the integrand of a light transport problem contains several discontinuities that largely invalidate the high-order convergence properties of quadrature methods.

For this class of problems, Monte Carlo methods are a more suitable and common

approach. Monte Carlo methods use non-deterministic quantities in order to estimate a deterministic result. For the problem of computing a given integral

$$I = \int_{\mathcal{X}} f(x)dx, \tag{2.34}$$

a Monte Carlo method *estimates* the value of integral I by combining random samples of the integrand function taken over the domain $\mathcal{X}$. Assuming, without loss of generality, that the integration domain has unit measure, a single estimator for the integral I can be constructed by creating a set $\mathcal{X}'$ containing random sample coordinates uniformly distributed over the domain of integration and averaging the integrand samples evaluated at these sample coordinates to find the expected value

$$I \approx \widetilde{I}_N = \frac{1}{N} \sum_{x \in \mathcal{X}'} f(x), \tag{2.35}$$

where $N = |\mathcal{X}'|$ is the number of samples used to compute the estimate $\widetilde{I}_N$.

Monte Carlo methods are called *robust* because they make almost no assumptions about the integrand in order to guarantee a specific order of convergence. The main disadvantage of Monte Carlo methods is that generally their convergence is slow, or presents lower accuracy depending on integrand features. The order of convergence of the estimator defined in equation 2.35 is $O(N^{-1/2})$. That means, for instance, that in order to reduce the approximation error of a Monte Carlo estimator by half, the number of samples required must be increased four times. However, this convergence rate is

independent of dimensionality and the presence of discontinuities.

Two important properties used to characterize Monte Carlo estimators are bias and consistency. *Bias* is the difference between the expected value of an estimator and the exact value of the integral. An estimator is called *unbiased* when its expected value has no bias, being exactly the value of the integral. A biased estimator is called *consistent* if its bias decreases towards zero as the number of samples increases [38]. The simple estimator described in equation 2.35 is unbiased, but more complex estimators, such as estimators based on adaptive sampling of the domain of integration, can introduce non-negligible bias on the estimate [56].

The error associated with an estimator is usually defined as the sum of the standard deviation and the bias. That means it may be worthwhile to use a biased estimator if the variance is reduced enough to compensate for the introduced bias. However, the introduction of bias makes the analysis more difficult than for unbiased estimators.

Improving Convergence

The Monte Carlo convergence rate comes from the fact that the variance of an estimator decreases linearly as the number of samples increases. Since the error of an estimator is determined by its standard deviation, that implies the error associated with a Monte Carlo estimator decreases at the same rate as $\sqrt{N}$ increases. A common approach to improve the convergence of a Monte Carlo method is to reduce the variance associated with its estimator.

Most Monte Carlo integration techniques can be separated into *blind Monte Carlo* and *informed Monte Carlo*. Blind Monte Carlo techniques assume no information about the integrand while informed Monte Carlo techniques define estimators based on previous information about the integrand. The most widely used techniques in computer graphics for variance reduction of Monte Carlo estimators are *stratified sampling* (a blind Monte Carlo technique) and *importance sampling* (an informed Monte Carlo technique) [39].

Stratified sampling is a variance reduction technique based on the subdivision of the integration domain into subdomains, called strata, and evaluation the integral of each subdomain separately (using typically one sample per strata.) The objective of stratified sampling is to provide a better distribution of samples over the integration domain, reducing the undesirable clumping of samples that commonly occur using uniform random sampling. Clumping of samples is undesirable because samples that are close together provide less information about the integrand than samples that are positioned far apart.

Using stratified sampling often increases the convergence rate of a Monte Carlo method. The variance obtained using a stratified sampling technique is never higher than the variance obtained using a uniform random sampling scheme. However, the improvement in the convergence rate of stratified sampling depends on the smoothness of the integrand and the dimensionality of the integration domain.

Mitchell [57] has presented a study of the effect of the smoothness of the integrand on the convergence of stratified sampling for two dimensional domains. The main results show that, on two dimensional domains, when the integrand is smooth, with continuous

and bounded first derivatives, the convergence rate can be improved to order $O(N^{-1})$. However, if the integrand is only piecewise continuous, the convergence rate is improved only to $O(N^{-3/4})$. If the integrand is highly discontinuous, the improvement achieved is negligible.

Regarding the domain of integration, stratified sampling is useful only when the domain of integration has a relatively small dimensionality, typically less than 4. The main issue is that the number of required strata does not scale well to the dimensionality of the domain, requiring N^d samples for a d-dimensional domain. Several techniques have been proposed to address the increase of the number of samples with respect to the dimensionality of the domain, including the N-rooks algorithm and our own approach, generalized stratification using Hilbert curves [58].

Unlike stratified sampling, importance sampling is an informed variance reduction technique that uses *a priori* information about the integrand in order to reduce the variance of an estimator. Importance sampling approaches the variance reduction problem using an *importance function* defined over the domain of integration. This importance function can be used to specify a nonuniform *probability distribution function* (PDF) $p(x)$ according to which the integrand is sampled. Choosing a suitable importance function can significantly reduce the variance of a Monte Carlo estimator [38].

Importance sampling is based on the use of a nonuniform PDF to select the set of samples that are used to compute the estimate. Using importance sampling, the integral

defined in equation 2.34 can be estimated by

$$\widetilde{I}_N = \frac{1}{N} \sum_{x \in \mathcal{X}'} \frac{f(x)}{p(x)} \tag{2.36}$$

where $p(x)$ is a probability distribution function defined over the domain of integration $\mathcal{X}$ and the N sample coordinates in $\mathcal{X}'$ used to compute the estimate are drawn according to the PDF $p(x)$.

Based on equation 2.36, it can be seen that in order to reduce the variance of an estimator, one must choose a PDF $p(x)$ as proportional as possible to the function $f(x)$. Choosing a PDF proportional to $f(x)$ yields a term $f(x)/p(x)$ close to a constant for all samples and consequently reduces the variance of the estimate. However, the PDF $p(x)$ must be chosen carefully. If the PDF used for a given estimate is very different from the function $f(x)$, importance sampling can yield a *worse* convergence rate than uniform random sampling since $f(x)/p(x)$ may have higher variance than $f(x)$ alone.

The importance sampling technique can be extended in order to handle several different estimators. Multiple importance sampling [59] is a technique that allows for the combination of different estimators, each one suitable under different circumstances, by assigning different weights to each estimator, or to each sample used by an estimator. This technique exploits the fundamental property that the expected value of a linear combination of estimators is the same as the linear combination of the expected values of each estimator. In fact, Veach [59] has proved that this linear combination is optimal

when weights are inversely proportional to the variance of each estimator.

It is possible to combine importance sampling and stratified sampling techniques. This can be done by drawing stratified samples over the integration domain and then distributing these samples according to a probability distribution function, using a standard inverse cumulative distribution function sampling. Both stratified and importance sampling techniques produce unbiased estimators, so that they can improve the convergence rate of an estimator without introducing artifacts. Since both techniques are unbiased, their combination, called *stratified importance sampling*, is also unbiased.

2.4 Algorithms for Light Transport

The rendering equation provides an elegant mathematical formulation of the light transport problem. However, it does not have a closed form solution, except for extremely simple cases. In order to compute a solution to the rendering equation, numerical integration methods must be used. A suitable approach to this problem is to evaluate the rendering equation using Monte Carlo integration.

Considering the hemispherical formulation of the rendering equation defined in equation 2.17, it is possible to approximate the integral by averaging samples of the integrand:

$$L(\mathbf{x}, \hat{\omega}_o) = L_e(\mathbf{x}, \hat{\omega}_o) + \frac{1}{N} \sum_{\hat{\omega}_i \in \mathcal{W}_\mathbf{x}} f_r(\hat{\omega}_i, \mathbf{x}, \hat{\omega}_o) L(h(\mathbf{x}, \hat{\omega}_i), -\hat{\omega}_i) \cos\theta_i \tag{2.37}$$

where $\hat{\omega}_i$ is a direction uniformly sampled over the hemisphere $\Omega_\mathbf{x}$ and $N = |\mathcal{W}_\mathbf{x}|$ is the

number of directional samples used in the computation.

Importance sampling can be included in the numerical solution of equation 2.37. Assuming that directions $\hat{\omega}_i$ are now sampled according to a PDF $p(\mathbf{x}, \hat{\omega})$, the final expression becomes:

$$L(\mathbf{x}, \hat{\omega}_o) = L_e(\mathbf{x}, \hat{\omega}_o) + \frac{1}{N} \sum_{\hat{\omega}_i \in \mathcal{W}_\mathbf{x}} \frac{f_r(\hat{\omega}_i, \mathbf{x}, \hat{\omega}_o) L(h(\mathbf{x}, \hat{\omega}_i), -\hat{\omega}_i) \cos\theta_i}{p(\mathbf{x}, \hat{\omega}_i)} \tag{2.38}$$

Most algorithms for computing global illumination solutions can be separated into two main categories: *ray-path* and *finite-element* based algorithms.

2.4.1 Ray-Path Based Algorithms

Ray-path algorithms use rays or paths as the base mechanism to transfer light over the scene, and most compute the global illumination solution using a variant of the Neumann expansion described in equation 2.26.

Distributed Ray Tracing

Ray tracing [60] is a common algorithm in computer graphics for computing specular effects. Traditional ray tracing works by tracing rays from a camera position[1], recursively performing specular reflections and transmissions until each path reaches a light source.

One of the first ray-based algorithms to account for indirect illumination was dis-

[1]The pinhole camera analogy is commonly used in computer graphics to describe the point of convergence from a set radiance samples that are composed to form an image.

tributed ray tracing [61]. Distributed ray tracing performs a direct implementation of equation 2.38 and can be seen a generalization of the traditional ray tracing algorithm. The main difference between a traditional ray tracing and distributed ray tracing algorithm with respect to equation 2.38 is that a traditional ray tracing algorithm uses only a small number of deterministically selected samples to estimate the incoming radiance field at a given surface point, usually one sample in the specular reflection and refraction directions, and one sample in the direction of each point light source. Distributed ray tracing, on the other hand, uses several randomly distributed samples.

Distributed ray tracing estimates the integral in the rendering equation for a given surface point generating a set of child rays, and then recursively evaluates the radiance arriving from these rays. In order to approximate the incoming radiance from each child ray, the same procedure is repeated up to a termination condition. After the incoming radiance from all child rays is acquired, the integrand samples are evaluated (incoming radiance multiplied by the BRDF and cosine term) and averaged. Figure 2.14 shows the geometry of a distributed ray tracing algorithm.

Although this algorithm computes an approximation of the indirect illumination in the environment (instead of using a constant term like in traditional ray tracing), it tends to generate an excessive number of child rays. Also, even using a large number of rays, the results can still contain significant approximation errors. These errors are more likely to appear on scenes containing bright lights occluded from direct viewing [37]. These artifacts are caused due to the fact that the contributions of bright lights are impor-

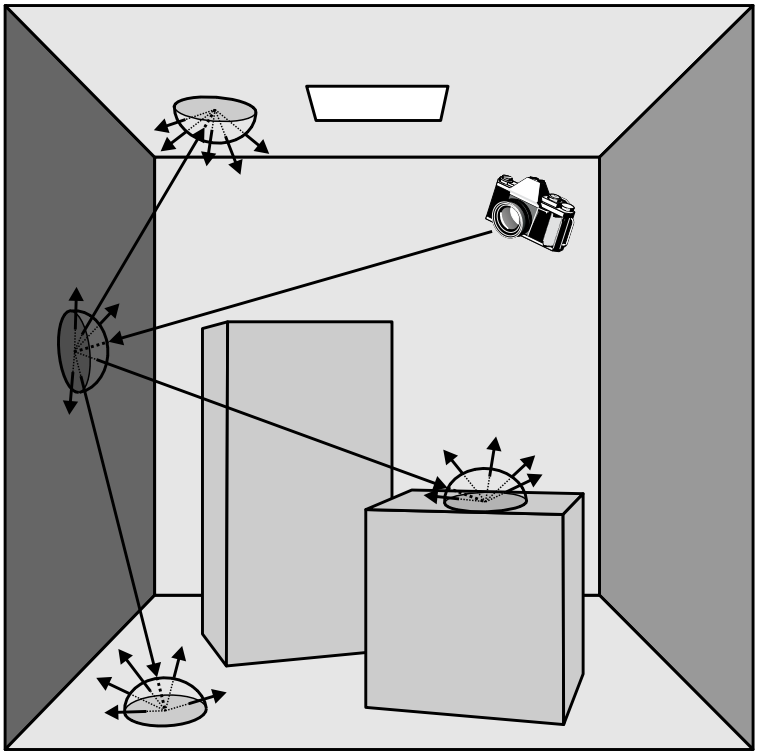


Figure 2.14: Distributed ray tracing evaluates the incident irradiance at each surface hit by generating new rays.

tant, but since their contributions depend on second or third bounces, the probability of accounting for these contributions becomes exponentially smaller.

When small bright light sources are present in a scene, a commonly used technique to improve convergence is *next event estimation* [37]. This technique estimates the direct illumination (radiance arriving directly from each light source) separately from the radiance resulting from indirect illumination. The direct illumination arriving from each light source, as well as the incident radiance from the indirect illumination, are accordingly weighted by their respective solid angles and the results added. Figure 2.15 describes the

next event estimation technique.

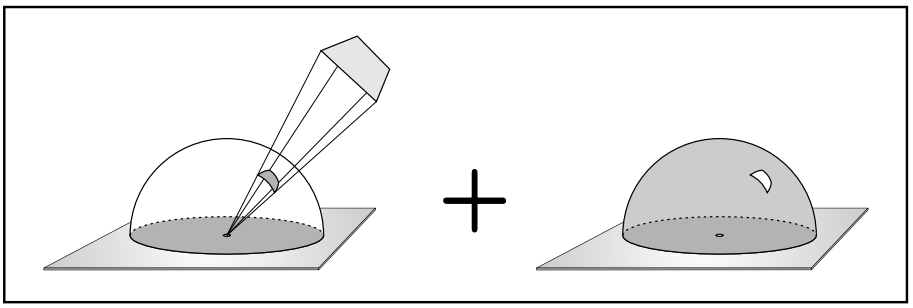


Figure 2.15: Next event estimation estimates the incident radiance coming from two disjoint domains separately, combining the results weighted by the corresponding solid angle measure.

Path Tracing

One of the major problems with the distributed ray tracing algorithm is the excessive number of rays used during an estimation. This happens mostly because the multiple-bounce operator is estimated by computing the incoming radiance for each ray bounce independently, a conservative approach. A more efficient approach to the evaluation of a multiple-bounce light transport operator uses a path formulation of the light transport problem [2].

The formulations presented in this section are based on the transport of exitant or incident radiance at individual bounce levels. This is equivalent to solving several two dimensional integral problems in a given order. A different strategy consists of defining a different domain of integration with higher dimensionality and selecting a smaller number of samples.

Path tracing is a technique that expresses the light transport problem using a path-space as domain of integration [50]. A path-space is defined as the space that contains all possible paths of any length over a scene. Using a path-space as the domain of integration allows the rendering equation to be expressed as

$$L(\mathbf{s}) = \int_{\bar{\mathbf{x}} \in \Omega^*} t(\bar{\mathbf{x}}) \mu(\bar{\mathbf{x}}) \tag{2.39}$$

where $\bar{\mathbf{x}}$ is a path of arbitrary length with endpoints at a camera sensor point $\mathbf{s}$ and a light source point. The domain of integration is given by the path-space Ω^*. The integrand is given by the function $t(\bar{\mathbf{x}})$ that defines the throughput of energy through a path $\mathbf{x}$. The throughput $t(\bar{\mathbf{x}})$ is the total radiance that leaves the light source at one endpoint and arrives at the camera sensor $\mathbf{s}$ on the other endpoint of a path $\mathbf{x}$. Also, $\mu(\bar{\mathbf{x}})$ defines a measure in path-space for a path $\bar{\mathbf{x}}$.

The main advantage of the path formulation is that paths with arbitrary length are treated as integrand samples. This enables a better control over the number of samples selected as well as the cost of individual samples.

Figure 2.16 describes the path tracing algorithm. The path tracing algorithm performs an integration of the incident radiance field over the path-space domain. The samples used for the numerical integration are selected by defining paths leaving a sensor to a light source. At each surface hit, the new path segment direction is selected randomly and the process is repeated until the path reaches a light source. However, there are

several optimizations that can be integrated into the path tracing algorithm.

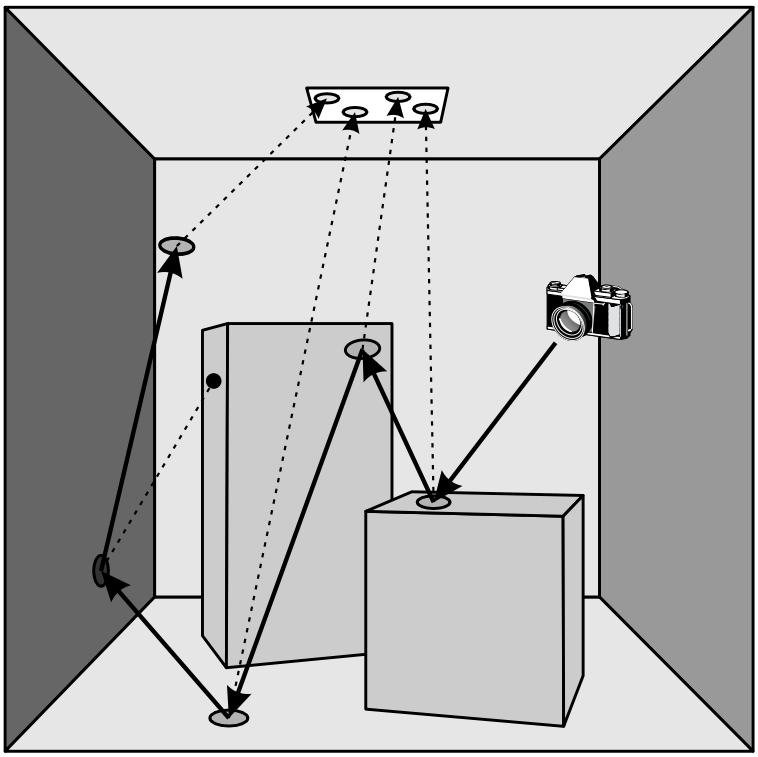


Figure 2.16: Path tracing estimates the incident radiance by averaging the contribution of several paths leaving the camera and arriving at the light source.

One optimization refers to the selection of the new direction while tracing a path from the camera. Instead of randomly selecting a direction and them multiplying the incoming flux by the BRDF and the cosine term, it is possible to sample according to the scattering probability function [52] at the hit point, averaging incident radiances.

Next-event estimation can also be used in a path tracing implementation by estimating the direct illumination from light sources at each surface hit. When handling point light sources this estimation can be done by simply tracing a shadow ray at each intersection

point between the path and a surface. If the direct illumination of area light sources are being estimated, tracing one or more shadow rays to surface points sampled over the light source is another simple optimization. Other alternatives include contour integration [62] or selecting one light sample at random at each bounce [63].

The path-space is an infinite dimensional space and therefore paths can be infinitely long. Since it is difficult to guarantee that no significant contribution will be added to a path, the simple truncation of a path may lead to significant bias on the final image. *Russian roulette* [64] is a technique that addresses this problem by *probabilistically* terminating a path. As length of a path increases, the relative contribution of the remaining path decreases geometrically. The Russian roulette technique takes this fact in consideration and terminates a given path randomly with probability $1 - P$. The weight P specifies the relative contribution of the path and can be given by the cumulative scattering probability function over the path

$$P = \prod_{i=0}^{n} s_r(\hat{\omega}_k, \mathbf{x}_k, \hat{\omega}_{k+1}) \tag{2.40}$$

where the path is defined by the sequence of directions $\{\hat{\omega}_0, \hat{\omega}_1, \ldots, \hat{\omega}_{n-1}, \hat{\omega}_n\}$ starting at the camera position.

The path tracing technique considerably reduces the computational cost necessary to achieve a image of comparable quality to that obtained with a distributed ray tracing algorithm. However, path tracing algorithms still have difficulties handling small light

sources when combined with specular reflection and/or refractions.

Light Tracing

Light tracing [3, 65] is a technique that attempts to address some of the limitations of distributed ray tracing and path tracing. Light tracing also integrates over a path-space, but using the exitant radiance field. Performing the integration using the exitant radiance field means that paths are generated starting at light sources, and bounce over surfaces in the scene until they reach the camera. The light tracing algorithm is a more natural approach for simulating the light transport problem, since in nature light is emitted from a light source, interacts with the environment, and eventually arrives at the camera. Figure 2.17 shows the basic idea of a light tracing algorithm.

The same optimizations presented for path tracing can also be used for light tracing. An almost mandatory optimization is the use of next-event optimization to make a light path reach the camera. This optimization is implemented by projecting the radiance reflected at a surface towards the camera position, accumulating the transported light at the corresponding view plane pixel. Even though light tracing addresses the problem of small and bright light sources in a scene, it has some severe limitations. Basic light tracing is not an efficient approach when the camera captures radiance coming from only a small part of the whole scene. Also, light tracing has difficulties handling surfaces that are highly specular. For classic global illumination scenes, it is generally accepted that a pure path tracing algorithm is more efficient than a pure light tracing algorithm.

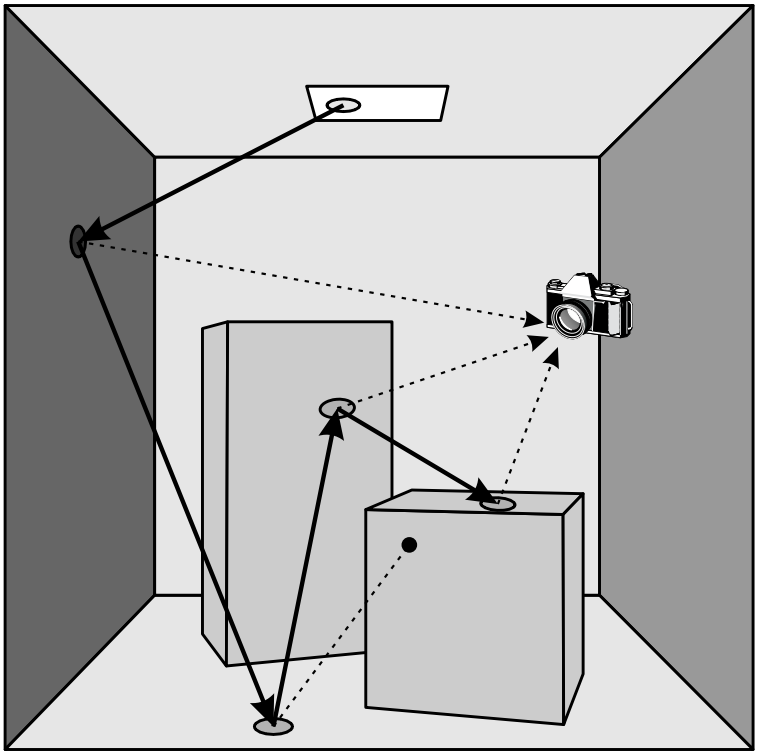


Figure 2.17: Light tracing algorithm transports light from light sources, projecting the reflected radiance towards the camera at each surface hit.

However, the light tracing approach is useful when combined with other techniques or when used to account for specific effects, like caustics.

Bidirectional Path Tracing

Bidirectional path tracing [4, 5] is a technique that addresses the main issues of both path and light tracing. This algorithm generates paths starting at the camera position, called *gathering paths*, as well as paths starting at light source points, called *shooting paths*. These two paths are then connected in the middle using a deterministic *shadow* ray in order to determine the contribution to a given pixel. Figure 2.18 illustrates the generation

of a path by a bidirectional path tracing algorithm.

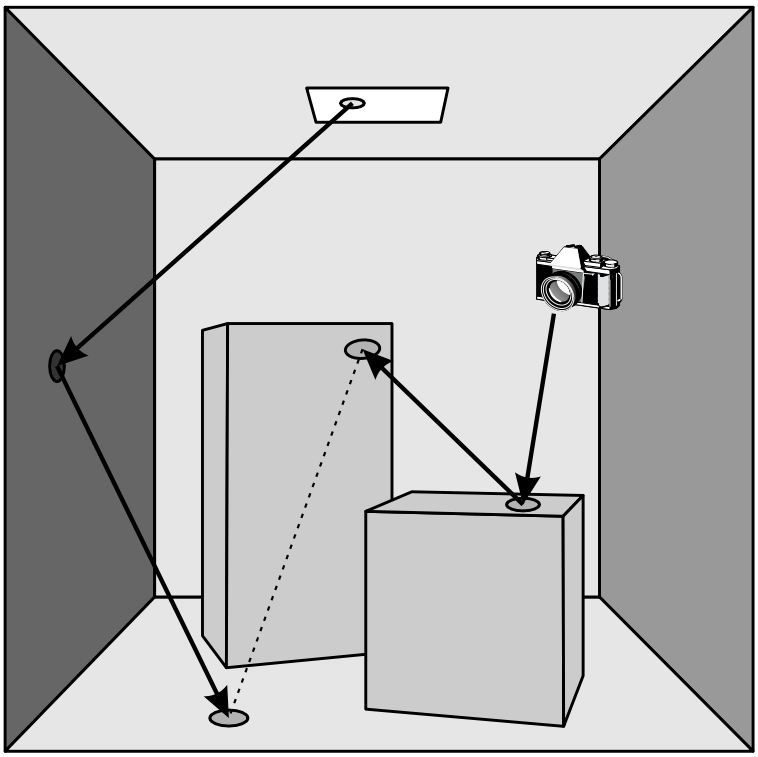


Figure 2.18: Bidirectional path tracing shoot rays from the camera and light sources, and connect them using a deterministic *shadow ray*.

However, it is important to observe that gathering and shooting paths use different radiance fields. A gathering path uses the incident radiance field while the shooting pass uses the exitant radiance field. In order to combine the contributions of both paths, the shooting path must be converted to a gathering path, or vice-versa [66].

A simple optimization that can applied the bidirectional path tracing is to account for the contribution of gathering and shooting paths at all path segments. Since only shadow rays are necessary in order to accumulate the contributions at multiple segments,

the cost to consider the contribution of the extra paths is relatively inexpensive. This optimization is illustrated in figure 2.19 for the same conditions on figure 2.18.

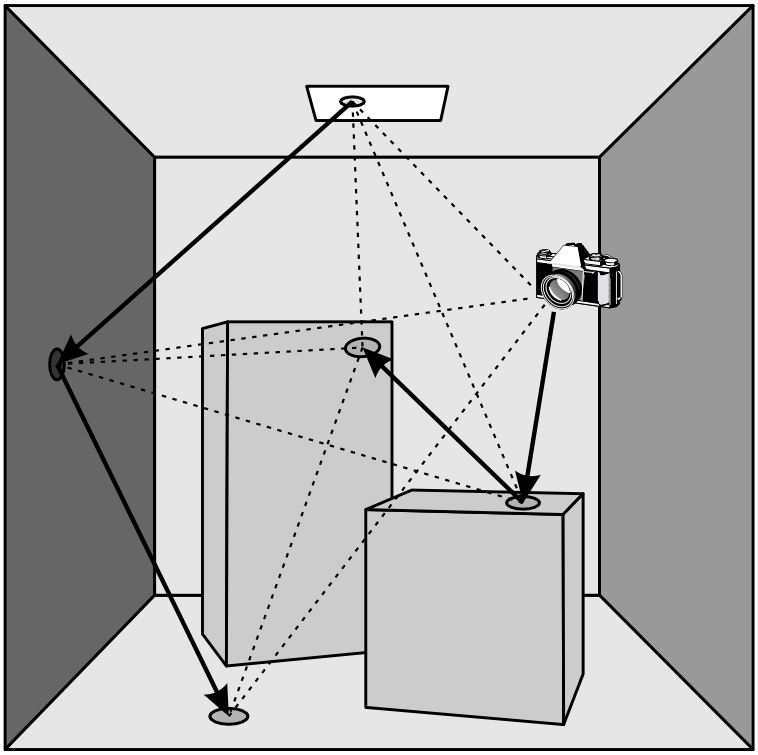


Figure 2.19: A bidirectional path tracing optimization consists of connecting all mutually visible shooting and gathering path endpoints.

Bidirectional path tracing algorithms have a major limitation. When shadow rays are evaluated between two specular surfaces, the BRDF is, with high probability, zero in the direction sampled. This situation is common when rendering illumination effects, like caustics seen through a lens. In fact, even the presence of glossy surfaces can reduce the efficiency of bidirectional path tracing by only accounting for marginal contributions along shadow rays. Finally, bidirectional path tracing is prone to *shot noise* due to the

$1/r^2$ term in the shadow rays, which is occasionally very large. This, however, can be dealt with by importance resampling [67]. In importance resampling, we must choose paths probalistically from the path population generated by bidirectional path tracing, rather than always taking all paths. The probabilties can be chosen proportional to the relative throughput of the paths.

A bidirectional algorithm for computing a global illumination solution that can handle difficult low-probability, high-thoughput paths is the *Metropolis light transport* algorithm [50, 68]. Metropolis light transport computes a global illumination solution by initially generating a set of light transport paths and then performing random mutations on these paths. The main idea of this approach is to account for paths that have higher contribution to the final image by generating a suitable probability distribution using a random walk. The Metropolis sampling scheme assumes that paths close to a path with a high contribution will have a similar contribution. The proposed mutations for individual paths allow for the local exploration of the path-space. Metropolis light transport algorithm performs well in difficult illumination situations, like environments with strong indirect illumination, small geometric holes, or a large number of glossy surfaces. However, it is relatively difficult to implement and the random walk can get stuck in local minima.

Photon Mapping

Bidirectional path tracing algorithms are able to account for a large number of illumination effects. However, when several images must be rendered from the same static scene, most of the incoming radiance over the environment remains unchanged. A more efficient approach in this case is to precompute and store the shooting paths on a first pass and, on a second pass, render the gathering paths for each image.

Based on this principle, Arvo [69] introduced a technique called *illumination maps* where illumination results are stored in object space using texture maps. Shirley [70] also used mesh polygonalization to store results from a set of illumation rays. In a technique called *photon mapping*, Jensen [6, 71] further extended these ideas using a more efficient data structure to store global illumination samples. Jensen also used these data structures to store both direction and power information, rather than just to accumulate irradiance information.

Photon mapping consists of a two-pass approach where, in the first pass, *photons* are shot into the scene and the flux information carried by them stored in a data structure called a *photon map*. These photons describe the effects of the shooting paths. On a second pass, gathering paths are computed and combined with the contribution from several nearby shooting paths simultaneously.

Photon maps typically store the photon information using a data structure organized as *kd*-trees [72] or a similar data structure. While evaluating the gathering paths, a k-nearest neighbor lookup is executed to select the shooting paths to be linked to the

gathering path. The photon information stored on photon maps usually includes incoming radiance, incoming direction, hit point, and the local surface normal at each hit point.

One advantage of photon mapping is that it decouples the illumination properties of a scene from specific surface parameterizations, enabling the use of arbitrary objects. Also, photon mapping can be easily extended to incorporate other illumination effects, like volumetric effects and participating media [73]. The main drawback of photon mapping is the large memory requirements for the storage of photon information. Also, determining the k-nearest neighbors in a large data set is a difficult problem to implement efficiently and requires highly optimized data structures. Finally, the photon mapping algorithm is biased, and a global illumination solution computed with too few photons can exhibit various artifacts, including light bleeding, darkening near edges, and missed illumination on small objects.

2.4.2 Finite-Element Based Algorithms

Finite-element algorithms use area elements as base primitives for transporting radiance and to compute the solution approximating the inverse of the transport operator, as described in equation 2.25. The most representative technique of this class of algorithms is the radiosity method.

The *radiosity* method for solving the light transport problem was first introduced by Goral et al. [1], and Nishita and Nakamae [74]. This approach is based on the discretization of the environment into a set of area elements, called *patches*. In the classic radiosity

approach, each patch is assumed to have a constant BRDF and consequently, constant outgoing radiance in all directions. We can therefore represent the outgoing radiance using units of radiosity (outgoing irradiance) instead of radiance. This assumption allows the area formulation of the radiance equation defined in (2.19) to be expressed as a problem with reduced dimensionality:

$$B_i = E_i + \rho_i \sum_{j=0}^{n} B_j F_{ij} \tag{2.41}$$

where B_i, E_i and ρ_i are respectively the radiosity, emissivity and reflectivity of patch i. The relationship between any two patches is defined by F_{ij}, known as the *form factor*. The form factor F_{ij} between two patches i and j describes the energy ratio that is transferred between these patches [36] and can be computed using

$$F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{\cos\theta_{\mathbf{x}} \cos\theta_{\mathbf{y}}}{\pi r_{\mathbf{xy}}^2} V(\mathbf{x}, \mathbf{y})\, d\mathbf{y} d\mathbf{x}. \tag{2.42}$$

Figure 2.20 shows the geometry of a form factor between two differential surfaces. The angles between the local surface normals at points $\mathbf{x}$ and $\mathbf{y}$ and the line joining them are given $\theta_{\mathbf{x}}$ and $\theta_{\mathbf{y}}$ and the distance between these two points is given by $r_{\mathbf{xy}}$. The function $V(\mathbf{x}, \mathbf{y})$ is the visibility function described previously.

Equation 2.41 can be solved by finding the solution of the following system of linear

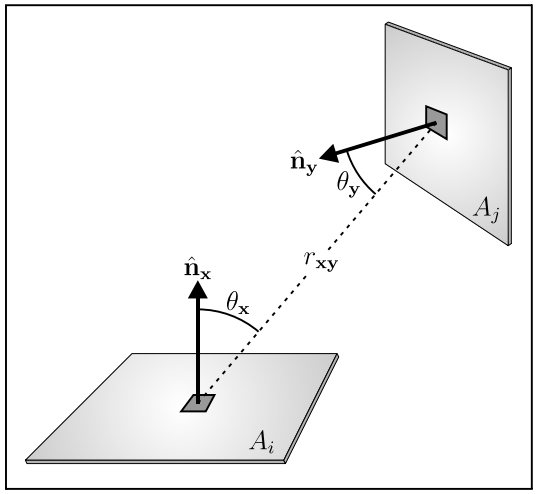


Figure 2.20: Geometry of the form factor between two differential areas. The form factor between two differential surface areas describes the ratio of energy that can be transferred between them.

equations:

$$\begin{bmatrix} 1 & -\rho_0 F_{01} & -\rho_0 F_{02} & \dots & -\rho_0 F_{0n} \\ -\rho_1 F_{10} & 1 & -\rho_1 F_{10} & \dots & -\rho_1 F_{1n} \\ -\rho_2 F_{20} & -\rho_2 F_{21} & 1 & \dots & -\rho_1 F_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -\rho_n F_{n0} & -\rho_n F_{n1} & -\rho_n F_{n2} & \dots & 1 \end{bmatrix} \begin{bmatrix} B_0 \\ B_1 \\ B_2 \\ \vdots \\ B_n \end{bmatrix} = \begin{bmatrix} E_0 \\ E_1 \\ E_2 \\ \vdots \\ E_n \end{bmatrix} \tag{2.43}$$

The main advantage (and disadvantage) of the radiosity method is its view independence. Once the radiosities (or irradiances) for all patches has been computed, the radiance values needed to render an image from a given viewpoint can be efficiently computed based on the assumption that all surface points on a patch have the same

reflectance properties. However, this property is also responsible for one of the major issues of finite-element techniques for light transport. In order to make the finite-element assumption valid, a correct discretization of the environment into a set of patches is necessary. However, environment discretization is a complex problem. If the environment is not sufficiently subdivided, visible artifacts occur because the uniformity assumption does not hold. If the environment is overly subdivided the computational cost for solving the radiosity equation increases significantly. Adaptive subdivision can be used to define a tighter subdivision scheme, but it is a computationally intensive operation.

Despite finite-element methods' issues, their advantages justifies the extensive use of radiosity algorithms in interactive walkthrough and architectural applications. Several techniques have been proposed in order to improve performance of the original radiosity method, such hierarchical radiosity [75] and importance-driven radiosity [76]. In addition, some methods have been proposed to handle patches with non-diffuse BRDFs [77]. Monte Carlo methods have been traditionally used to approximate form factors between surfaces. However, it is possible to use Monte Carlo methods not just to approximate form factors, but to solve the radiosity equation as well.

Shirley [78] has presented a technique for combining deterministic and probabilistic methods in order to solve the radiosity equation. Neumann further extend this technique using the *stochastic radiosity* method [79]. The stochastic radiosity method solves the radiosity equation by randomly selecting a fixed number of shooting patches at each iteration step and then only using these patches to transfer power to the rest of the

scene. Patches are selected with probability proportional to their shooting power. The shooting power of each selected patch is increased in order to represent the emitted power of all patches. The stochastic radiosity method has been extended by Neumann et al. [80] with the *stochastic ray method*. Instead of dividing the total power from the environment to a reduced number of patches to transfer power to the environment, the stochastic ray method divides the total power into a large number of rays. Each one carries the same amount of power and a number of rays is distributed to each selected patch proportionally to its fraction of the total power. These rays are then shot towards the environment and their reflected power is stored at the receiving patches. The procedure for shooting rays from a patch is done by selecting a random position on the patch and a random direction and then finding the closest patch along this ray. The main advantage of the stochastic ray method for solving the radiosity equation is that there is no need to explicitly compute and store form factors. The throughput between surfaces is instead stochastically approximated during the ray shooting procedure. Since the storage required for the form factor matrix is quadratic in the number of patches in the scene, this can lead to significant scalability and performance advantages.

2.5 Graphics Hardware

Several recent advances in graphics hardware architecture have driven the use of GPUs for general purpose computation. First, studies show that the computational power of graphics hardware processors is increasing at a rate significantly higher than general pur-

pose processors [81]. Second, high-performance graphics hardware has become cheaper and more accessible to personal computers. Third, graphics hardware has become increasingly more programmable, instead of just being configurable. The combination of these three key elements has been motivating researchers to use graphics hardware in order to improve the performance of applications not necessarily related to computer graphics, such as simulations, scientific computing and signal processing. Graphics hardware has also been used to solve computer graphics problems that were usually implemented using the CPU to perform most of the computation. One of these problems is global illumination.

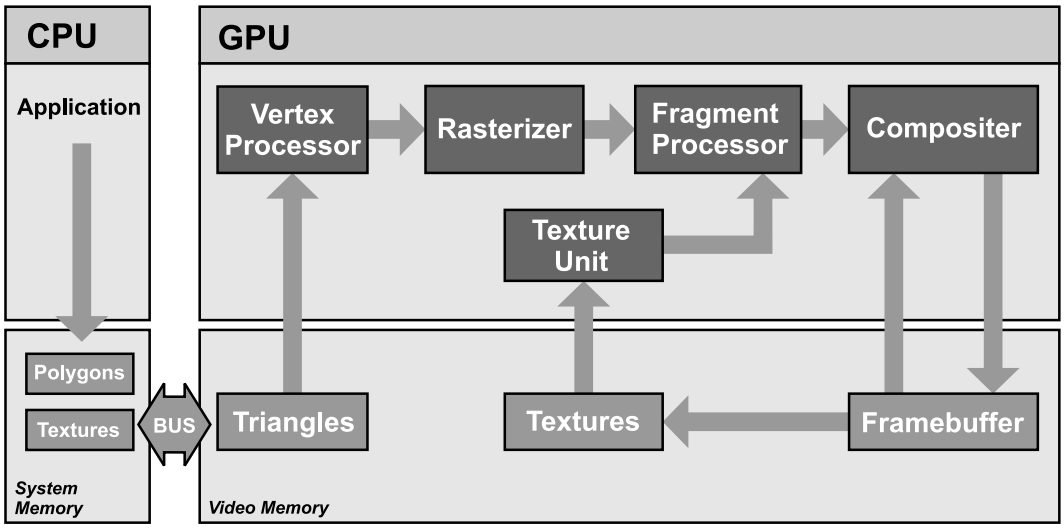


Figure 2.21: Rendering pipeline present on current graphics hardware.

This section aims to describe how the rendering pipeline is organized on most of current graphics cards. Based on the diagram shown in figure 2.21, we can describe the flow of information between CPU and GPU as follows. Initially, the rendering application run-

ning on the CPU transfers geometric data and texture information from system memory to video memory. When compared to other data transfers in the graphics hardware, this is a relatively slow operation. The uploaded geometry is usually composed of triangles. When this is not the case, it is usually processed by the rendering API into a set of triangles. Vertices composing the geometry are fed to a *vertex processor* that executes a *vertex program* using information attached to a given triangle vertex. Information is usually attached to vertices using texture coordinates or color data, but it is also possible to define a limited number of global parameters for all vertices. Processed vertices are then sent to a rasterizer module that assembles a sequence of vertices into triangles and performs its rasterization in screen space. The rasterization produces a set fragments (one or more per pixel) that are sent to a *fragment processor*. Each fragment has also a number of associated attributes that are computed by the vertex program and ratio-linearly interpolated over the triangle using the output attributes from each vertex. The fragment processor then executes a *fragment program* for each fragment using the data passed as fragment attributes. The resulting color data is finally sent to a compositor module that performs compositing and visibility operations and sends the final fragment to the framebuffer.

Some details that can be added to this overall description are that fragment processors have access to textures stored on texture units. Textures can be basically treated as indexable arrays of pixels, called *texels*, with added support for interpolation and filtering. Current graphics hardware has a limited number of textures units simultaneously available

to the fragment processor. Some vendors also allow access to textures by the vertex processors. The framebuffer contents can also be transferred to a texture and used by a fragment processor; this feature is supported on all graphics cards. However, most current graphics hardware supports directly rendering to a texture using a specialized buffer, called a *p-buffer*. This is still a relatively slow operation, even though it is faster than copying the framebuffer to a texture. However, a new API extension has been proposed for OpenGL, *frame buffer objects*, that will unify memory management for the framebuffer and textures and will make rendering into textures much more efficient.

Current GPUs normally have more than one vertex and fragment processor, allowing for the processing of several vertices and fragments in parallel. Also, registers available in these processors are tuples of four elements and can operate on the elements of these tuples in parallel too. The precision of current graphics hardware has also improved significantly in the last few years. Vertex and fragment processors support internal 32-bit floating-point operations. Several of these operations are specialized for graphics applications, for instance, dot product and reciprocal square root. Also, single and half floating-point precision is available for elements of textures and p-buffers.

The graphics hardware capabilities that are of interest to implement our light transport algorithm include shadowing and environment mapping, BRDF rendering using multiple light sources, and floating-point buffers.

2.5.1 Projective Texture Mapping

Projective texture mapping is a technique described by Segal et al. [82] that allows for the information stored in a two dimensional texture to be projected from a given point onto the environment, like a slide projector. Projective texture mapping differs from standard texture mapping mainly on how the texture coordinates are assigned to individual vertices. On standard texture mapping, texture coordinates are explicitly assigned to each vertex, while on projective texture mapping texture coordinates are computed based on a projection point. The projective texture coordinates are assigned to each vertex so that vertices collinear to the projection point have the same texture coordinates.

The texture coordinates can be computed using the vertex coordinates together with the projection position and orientation. Considering that the projection position and orientation are represented by a transformation matrix T responsible for transforming a vertex $\mathbf{p}$ to the projection coordinate system, the texture coordinates (u, v) can be computed by

$$u = \vec{\mathbf{v}}_x / \vec{\mathbf{v}}_z \qquad\qquad v = \vec{\mathbf{v}}_y / \vec{\mathbf{v}}_z \tag{2.44}$$

where $\vec{\mathbf{v}} = T\,\mathbf{p}$. Figure 2.5.1 describes an example of projective texture mapping. In this example, the points $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$ have the same texture coordinates. The most common applications of projective texture mapping include effects like virtual slide projectors and shadow maps.

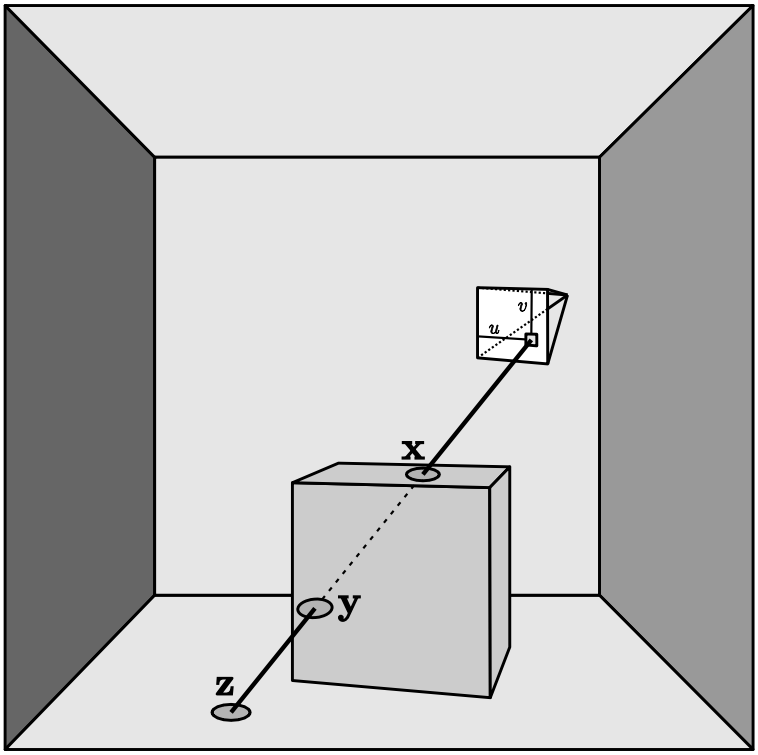


Figure 2.22: Example of projective texture mapping. Points **x**, **y** and **z** have the same texture coordinates (u, v).

2.5.2 Shadow Mapping

Shadow mapping is a technique proposed by Williams [83] that uses precomputed visibility information with respect to a point light source to determine the presence of a fragment in a shadow region. This technique uses a shadow texture to store the precomputed visibility information which consist of distances from the light source position to the closest point in the direction defined by the texture's texel.

Shadow mapping is a two-pass approach where, in the first pass, the depth information of the environment with respect to the light source is acquired and stored in a shadow

map texture. On a second pass, when a scene is being rendered from the camera point of view, the distance of a given fragment to the light source is computed and compared to the depth information stored in the shadow map texture. The texture coordinates used to look up this depth information can be computed using projective texture mapping. If the depth information stored in the shadow map is smaller than the distance from the fragment to the light source, the fragment is shaded as being in shadow. Figure 2.23 shows a situation where a fragment is shadowed and another where a fragment is lit.

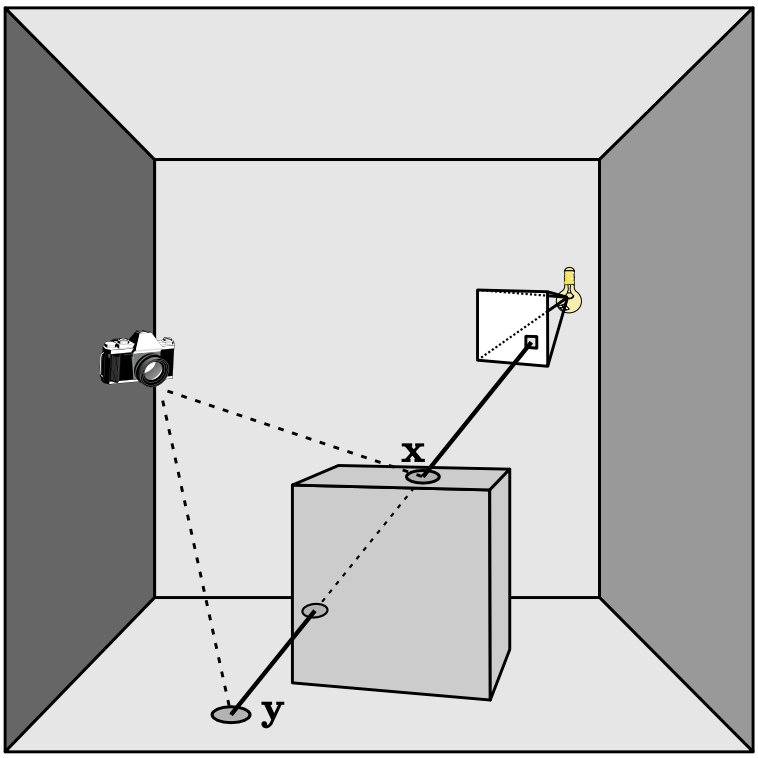


Figure 2.23: In a shadow mapping algorithm the distance from a light source to a given fragment is compared to the distance from this light source to the closest surface point (in the same direction to the fragment). Point **x** is lit since these distances are equal while point **y** is shadowed.

The main advantage of shadow mapping is its low dependency on the scene complexity, since the visibility information for the shadowing test is precomputed. Another advantage is that it can be easily integrated into a rendering engine. However, a disadvantage of shadow mapping is susceptibility to aliasing artifacts, especially when the shadow map has low resolution. Also, it is important that a shadow map covers the whole region where the corresponding light source can cast illumination.

Several extensions to shadow mapping have been proposed in order to reduce its deficiencies. Perspective shadow mapping [84] reduces aliasing artifacts by acquiring the depth information in an optimized device space coordinate system. Also, different shadow mapping parameterizations can be used to account for hemispherical or omnidirectional shadows [85]. In chapter 4 we introduce a new approach to these problems, based on the use of pyramid and octahedral parameterizations.

2.5.3 Environment Mapping

Environment mapping is a technique that approximates the incident radiance at an object and is typically used to simulate reflections of an environment without explicitly computing reflection rays. If an object is significantly smaller than the surrounding environment, then the radiance arriving from a given direction is approximately the same for all surface points of the object. In other words, for all surface points $\mathbf{x}$ belonging to a object with

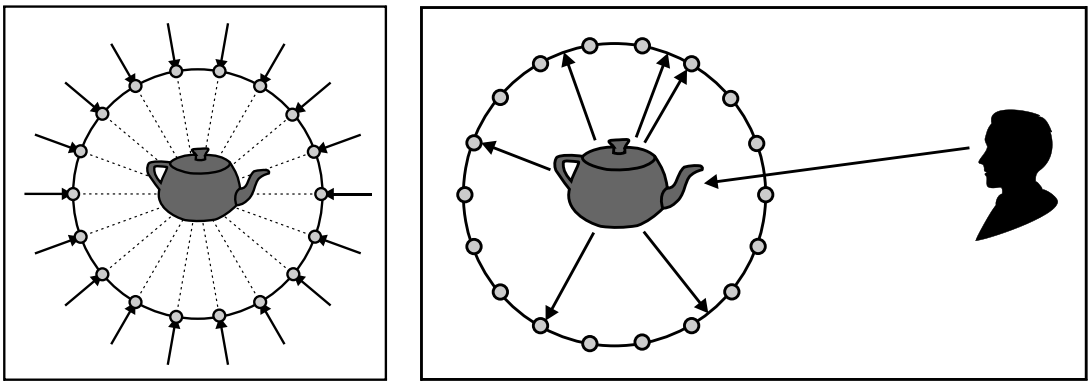
geometric center $\mathbf{c}$, the incident radiance field $L^*(\mathbf{x}, \hat{\omega})$ can be approximated by $L^*(\mathbf{c}, \hat{\omega})$:

$$\forall \mathbf{x} \in \mathcal{O} : L^*(\mathbf{x}, \hat{\omega}) \approx L^*(\mathbf{c}, \hat{\omega}) \tag{2.45}$$

where $\mathcal{O}$ is the set of all surface points of the object being handled. This means that, for the points in $\mathcal{O}$, the incident radiance is dependent only on the incident direction $\hat{\omega}$ and can be treated as a two dimensional field. When the incident radiance can be approximated as a two dimensional field, it can be conveniently represented using a two dimensional texture.

Rendering using environment maps consists of two stages. In a pre-processing step radiance field samples are acquired and stored in a texture. This can be done either in real-time rendering, using an off-line rendering system, or even by processing panoramic photographs of a real environment. When performing the rendering of an environment mapped object, this texture is used to query information about radiance arriving at the object. This technique is usually used to render approximated specular reflections, since the reflected direction can be computed based using the incident direction and the local surface normal. These two stages are shown in figure 2.24.

The radiance field incident to the object's center has a spherical topology while two dimensional textures are usually stored using a planar topology, so a parameterization mapping is necessary. Different parameterizations have been proposed to implement environment maps on graphics accelerators, including spherical [86], cube [87] and parabolic



(a) (b)

Figure 2.24: Environment mapping: (a) In the acquisition stage radiance samples are stored in a textures. (b) During the rendering stage, an environment map is used to approximate the incident radiance at a given object.

mappings [88, 89].

Spherical environment maps were introduced by Haeberli [86]. This parameterization is based on a two dimensional projection of a mirrored sphere. Even though most graphics accelerators natively support spherical environment maps, they have severe view-dependence limitations and are valid only for one map orientation.

Cube environment mapping [87] overcomes the limitations of sphere mapping by using a cube instead of a mirrored sphere to represent the incident radiance field. Cubemaps store the incident radiance in six separate two dimensional textures, one for each face of the cube. Cubemaps do not have the view dependence limitation of spherical mapping and are supported on all current graphics hardware. However, cubemap implementations on current graphics hardware is not as flexible as the implementation available for two dimensional textures. Some of the most important limitations include the lack of floating-

point support, interpolation, and filtering. Recently, some vendors started to have some limited support to floating-point cubemaps, and better support is likely in the future, since floating-point values are useful for representing the high dynamic range of illumination present in real environments.

Parabolic environment maps are an alternative parameterization proposed by Heidrich [88, 89] where the radiance arriving from an hemisphere is projected onto a circular domain using a paraboloid as base primitive. This environment mapping technique can then use two separate standard textures to represent a environment map. The parabolic texture coordinates can be computed by

$$u = \frac{\hat{\mathbf{v}}_x}{\hat{\mathbf{v}}_z + 1} \qquad v = \frac{\hat{\mathbf{v}}_y}{\hat{\mathbf{v}}_z + 1} \tag{2.46}$$

where $\hat{\mathbf{v}}$ is the vector from $\mathbf{c}$ in the direction of $\mathbf{x}$ and is required to have unit length. This parameterization is equivalent to using the paraboloid

$$f(x, y) = \frac{1}{2} - \frac{(x^2 + y^2)}{2} \tag{2.47}$$

as the base primitive for the environment map projection. Figure 2.25 shows the basic mechanism used to compute the texture coordinates using the parabolic projection.

One immediate benefit of parabolic environment maps is the use of standard two dimensional textures, which have better support on current GPUs. Also, parabolic maps have a more uniform sampling rate than cubemaps. However, the main disadvantage of

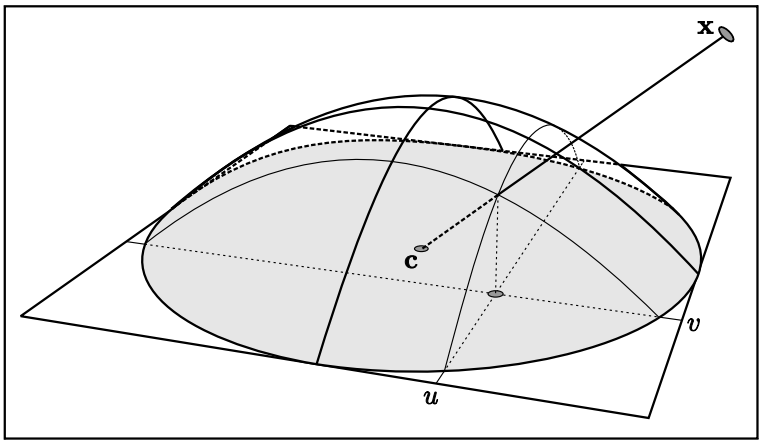


Figure 2.25: Geometry of parabolic projection: the intersection of the line segment given by **x** and **c**, and paraboloid given in equation 2.47 is projected in the base plane to obtain the texture coordinates (u, v).

the parabolic projection is that it is a non-linear projection, considerably reducing its usability on applications with dynamic environments, as we will explain shortly.

One major issue for the use of environment mapping on dynamic environments is the efficiency for generation of the image map. Cubemaps require the rendering of six separate images, one for each cube's face. The cubemap implementation on current GPU requires each image to be acquired on separate rendering contexts. Changing rendering contexts is a costly operation on current GPUs and can significantly reduce the overall performance of the application. Also, combining two or more cubemaps in one bigger cubemap is not a simple task and requires elaborate and costly texture lookup adaptations, limiting the number of cubemaps that can be used simultaneously by a fragment program.

Parabolic maps, on the other hand, use standard two dimensional textures to store the information from one hemisphere. This allows the information from a complete sphere

of directions to be represented on two square textures. Two dimensional textures can be easily packed together to enable the use of many parabolic textures simultaneously. However, the parabolic projection is non-linear, mapping lines from world space into parabolic arcs on texture space. This non-linearity can introduce severe artifacts when the solid angle subtended by a primitive is large. This situation occurs frequently in normal rendering and walkthrough scenes, when large primitives are being used or when the camera is moved close to an object. Also, parabolic projection maps a hemisphere onto a circle, and therefore uses only approximately 78.5% of the available texture memory. We present a new environment map scheme in chapter 4 that addresses some of these limitations.

2.6 Summary

This chapter presented the main background information required to understand the pencil light transport method. We initially presented the main radiometric concepts, followed by a general description of the theory for light transport. We briefly discussed Monte Carlo methods, a common method for numerical evaluation of the light transport equation. We also described classic algorithms used for simulation of light transport. We ended the chapter by presenting the graphics hardware rendering pipeline and the some hardware-supported techniques that can be used in a pencil light transport implementation on a GPU.

Chapter 3

Related Work

Global illumination is a research area that has been undergoing constant and active development in the last two decades. Many techniques have been developed to generate and visualize scenes including global illumination effects. Interactive rendering has been one of the main goals of many of the techniques developed recently. Most of the techniques that aim for interactivity perform incremental updates on special data structures containing precomputed illumination samples.

These methods are known as *caching schemes* and exploit temporal and spatial coherence of the radiance field. These methods rely heavily on an assumption that the observed radiance field does not change considerably between consecutive frames when a camera or an object moves. This class of methods is especially suitable for interactive walkthroughs with small changes in the geometry of the scene. Caching techniques rely on the use of special data structures to store previously computed high-quality illumina-

tion samples. However, large changes to the lighting conditions or to the geometry of the environment can invalidate most of the cache content, requiring the recomputation of a considerable number of new global illumination results or an acceptance by the user of inaccurate results for a period of time.

With the advent of programmable GPUs, it has been a natural trend to use the graphics hardware capabilities to accelerate the computation as well as the visualization of global illumination solutions. Techniques for handling the balance between CPU and GPU usage vary based on application's interactive requirements and the illumination effects they prioritize. In general, they can be classified according to the amount of computation that is performed on the CPU and on the GPU. Based on the ratio of CPU/GPU usage, techniques can be classified as *CPU-based* or *GPU-based* techniques.

3.1 CPU-based Methods

CPU-based methods compute most of the global illumination solution using the CPU. These techniques are also referred as *interactive methods* since their main focus is to produce global illumination solutions at interactive rates, independently of the graphics hardware usage. A system is generally considered interactive if it can perform rendering updates at 1 Hz or more [90]. However, there are also several CPU-based techniques that defer part of the global illumination computation to the visualization stage and take advantage of the graphics hardware capabilities.

The first global illumination methods to aim for interactive visualization of general

scenes were radiosity based methods [1, 91]. These methods performed the computation of the global illumination solution using the CPU and visualize the solution using standard graphics hardware. However, objects are assumed to have diffuse reflectance and emittance and the changes in the environment require the global illumination solution to be recomputed.

Most of interactive CPU techniques are based on the use of caching schemes. Several caching techniques have been proposed and differ mostly according to the information that is cached, the domain where illumination samples are computed, and the interpolation schemes used to approximate the observed radiance field between computed samples.

3.1.1 Interactive Radiosity

Several methods have been developed based on an adaptive hierarchical discretization of the environment that can compute global illumination solutions at interactive rates. Drettakis and Sillion [18] described a technique that augments the hierarchical radiosity clustering approach by defining a *line-space hierarchy*. This hierarchy can be used to identify link modifications caused by movement of objects in the scene, or to rebuild subdivision levels when the radiance gradient changes sufficiently.

A large number of different techniques have been described to overcome the non-diffuse assumption of early radiosity methods at interactive rates. Stamminger [92, 93] has proposed enhancing the hierarchical radiosity methods using *illumination samples* to allow glossy illumination effects. The main idea of this technique is to store illumination

samples on a given receiver containing the incoming direction and irradiance coming from a sender element. During the rendering process these illumination samples are then queried to determine the overall glossy reflection toward the camera.

Granier et al. [94, 95] used a similar idea to integrate particle tracing and hierarchical radiosity to account for diffuse and non-diffuse radiance transport effects, such as caustics and glossy reflections. The integration is done by tracing particles containing the non-diffuse component of the radiance transfer and using hierarchical radiosity to account for the transfer of the diffuse component of the illumination. This integration can be done at a small overall cost since information used by the hierarchical radiosity system can be used to guide the particle tracing operation.

3.1.2 Image Space Caching Methods

Image space caching methods are based on the reuse of global illumination elements sampled in image space. The *render cache* [9] is a technique that exploits the temporal and spatial coherence of sequences of camera motions as well as small changes in the illumination conditions of the scene. The render cache technique is based on the idea of decoupling the render engine from the display process, allowing the visualization module to approximate most of the scene illumination using information previously computed and stored in the cache. Typical information stored on a render cache element are the 3D position, color, object ID, and age of a sample. This information is used by the image generation process to provide a suitable approximation for the current view. The

cached information is managed by caching heuristics to determine the creation of new illumination samples as well as the deletion of unnecessary or inaccurate ones.

The process of image generation consists of querying shading information from the render cache and projecting it onto the current view plane. Depth culling using a modified z-buffer is used to approximate correct occlusion. Projected illumination samples are interpolated in order to provide a dense image without the gaps that naturally arise from the sparse illumination data set stored in the render cache.

Several improvements to the render cache method have been proposed [96], including the use of split projections, tiled z-buffers, predictive sampling and adaptive prefiltering with variable kernel footprints. Using split projections and tiled z-buffers improves memory coherence and allows for better parallel implementation. Predictive sampling allows the request of shading data for specific regions before they become visible. Variable kernel footprints for filtering of shading samples helps to reduce artifacts caused by sparse shading data at view plane.

Bala et al. [10] have presented a technique, called *edge-and-point image rendering*, that addresses most of the render cache artifacts, including blurring of sharp features caused by shading discontinuities, such as shadows and silhouettes. This is accomplished by analytically computing the locations of these shading discontinuities and using these boundaries to guide the interpolation process. Sparsely distributed shading samples are combined using an edge-constraint interpolation scheme, accounting for the efficient representation of perceptually important discontinuities.

3.1.3 Object Space Caching Methods

A different approach to caching is to organize the caching data structure so that the illumination samples are stored in object space. *Irradiance caching* is a technique proposed by Ward et al. [11] and is based on the fact that the irradiance due to indirect illumination has a smooth distribution over surfaces. The irradiance caching method computes a set of high-quality samples distributed over the scene and performs a interpolation for the inbetween values.

Several improvements have been proposed for the irradiance cache approach. Zaninetti et al. [97] proposed the use of *light vector caching* which stores a set of radiance samples instead of irradiance, allowing the rendering of glossy objects. Zanineti proposed the use of *kd*-trees for storing the radiance samples. However, this data structure produces several artifacts during interpolation. Crespin and Peroche [98] proposed the use of five dimensional grid to store the radiance samples. While the five dimensional data structure reduces the artifacts due to interpolation, it requires an expensive interpolation operation and has a costly first pass.

The *Tapestry method* proposed by Simmons and Séquin [12] provides interactivity by defining a three dimensional mesh over the environment that is then used to select and compute high-quality samples. This mesh is projected over a sphere around the camera and is used to determine regions on the view plane that are undersampled. When a region is determined as undersampled, new illumination samples are introduced and the mesh is refined accordingly. The mesh is created using a Delaunay triangulation allowing for a

more robust and efficient update procedure.

Tole [13] has proposed a extension to the render cache technique to reduce reprojection artifacts. The *shading cache* technique stores the illumination information on a mesh defined in object space that is progressively refined. Even when using a sparse set of radiance samples, the use of a shading mesh during the interactive rendering of the scene provides a dense radiance field approximation without gaps. The shading cache approach uses only one mesh to compute an approximation of the correct global illumination solution. This may lead to oversampling or limited reutilization of the cached illumination samples. Fournier and Peroche [99] further extend the shading cache method by creating a multiple mesh representation of the environment to store illumination samples. Meshes are created to store the direct diffuse, direct specular and indirect diffuse components of the global illumination solution. These meshes are progressively and independently refined. Also, a mesh is created to store the direct irradiance for each light source as well as for the overall indirect irradiance.

3.1.4 World Space Caching Methods

Some techniques organize the illumination samples independently of object parameterization and without any explicit dependency on a given view plane. Illumination samples are organized either in free space or by accounting for the whole environment's geometry. These methods are called world space caching methods.

Ward proposed an approach similar to the irradiance cache method, named *Holodeck*

ray caching [14]. Unlike the irradiance cache method, the Holodeck cache uses a four dimensional data structure organized to store rays belonging to the same beam at each entry. This allows that partial contributions from path segments to be more efficiently reused.

Greger [15] also proposed a natural extension to the irradiance caching method named the *irradiance volume*. The irradiance volume approach uses a volumetric approximation of the irradiance distribution function instead of selecting irradiance samples over surfaces. However, the irradiance volume is defined as a five dimensional function. This is due to the fact the irradiance field (but not the radiance field) is a smooth and continuous function directionally, but it can be discontinuous spatially. In order to solve this problem, a double interpolation scheme is performed. In a first stage, the irradiance is interpolated directionally and on a second stage it is interpolated spatially.

The irradiance volume technique has been enhanced by Nijasure et al. [100] by using a spherical harmonic representation of the incoming radiance at grid elements. Using this representation makes the technique more suitable for hardware implementation and considerably reduces the amount of memory required.

A similar idea is exploited by Bala et al. [16, 101] with the use of the *radiance interpolant* technique. This approach stores radiance samples using a four dimensional hierarchical data structure called a *linetree*. A linetree is basically a generalization of an octree to four dimensions. The use of a linetree to store radiance samples also enables the computation of conservative bounded-error measures via interval analysis techniques.

These error measures can then be used to adaptively refine the linetree by selectively adding more radiance samples. This technique can be used to guarantee interpolation errors within a user-specified error bound. This technique was further extended by using a five dimensional ray space hierarchy [102] to update radiance interpolants, enabling the clustering of a group of rays that are affected by changes on specific three dimensional regions.

Dmitriev et al. [17] has proposed an alternative clustering method for radiance samples, called *Selective Photon Tracing*. Selective Photon Tracing uses quasi-Monte Carlo[1] sampling sequences to cluster photons. The clustering is based on the fact that photons traced using similar n-dimensional Halton sequences follow similar paths because of the periodicity property of quasi-Monte Carlo sequences. Based on this fact, invalid photons with similar paths along an environment can be efficiently identified and updated using a relatively small number of selected photons.

3.2 GPU-based Methods

Recently, graphics hardware has become powerful and generically programmable enough to enable the implementation of several stages of the global illumination solution exclusively using the GPU. Graphics hardware has been successfully used to accelerate several traditional global illumination techniques, such as radiosity, ray tracing and photon map-

[1]Quasi-Monte Carlo methods evaluate integration problems using *low-discrepancy sequences* [38, 103].

ping. On the other side, new techniques have been developed that use specific features of graphics hardware, especially texture mapping and projection, vertex and fragment shader programmability, and fast visibility tests.

3.2.1 Radiosity Methods

The most computationally expensive component in the radiosity equation is the evaluation of the form factors between surfaces. Rushmeyer [104] proposed the use of cube projections in order to compute form factors between a differential surface area and all patches on the scene. This formulation is based on the hemicube method for form factor computation, but different projections, like parabolic or stereographic projections, can also be used for computing form factors [105].

Carr et al. [106] proposed a technique for solving a radiosity system on the GPU using a multiresolution mesh [107] atlas to parameterize the scene geometry into a two dimensional texture. Based on this two dimensional representation of the scene, the radiosity equation can be solved using the Jacobi iteration method. However, form factors between patches on the scene are still computed on the CPU in advance, and for real scenes, computation of the form factors, not solution of the matrix, is the dominant cost. Carr et al. also described how this approach can be used to simulate subsurface scattering on (rigid) translucent objects in real-time. This is an interesting application, since the lighting falling on an object is allowed to change dynamically, but as long as the object does not deform, it internal form-factor matrix does not need to be updated.

Coombe et al. [105] also used the idea of texels as radiosity elements. However, Coombe et al. described how to implement this method combined with an adaptive subdivision scheme. Adaptive subdivision is done based on radiosity gradient discontinuities and using a multiple-texture tree to account for geometry refinements. The radiosity solution is then obtained by executing a hierarchical progressive radiosity solver implemented on the GPU.

A different approach to the solution of the radiosity equation has been taken by the *instant radiosity* method [28]. The instant radiosity technique approximates the solution of the light transport equation by distributing a set of point light sources over the environment. The distribution of these light sources is done by generating paths along the scene, and then creating a point light source at the end of each ray segment. At each surface hit, the power of the light source created is attenuated by the surface's reflectivity. The final image is then generated by rendering the environment using all light sources created. However, this algorithm can have severe artifacts if the true intensity distribution of the light sources is used, including a $1/r^2$ term. In the original implementation, these artifacts were avoided by the fact that the hardware clamped large output values to a fixed range. However, this leads to a biased solution.

3.2.2 Ray Tracing

Ray tracing is another popular global illumination algorithm suitable for GPU implementation due to its high potential for parallelism. Purcell et al. [31, 108] described a method

to implement a ray tracing algorithm on graphics hardware by treating a GPU as a stream processor [19]. Purcell showed that the ray tracing algorithm can be implemented using a sequence of kernels executed over streams. These kernels include an eye-ray generator, grid transverser, ray-triangle intersector, and shader.

The eye-ray generator kernel computes a ray leaving the camera position in the direction of each pixel. Purcell's implementation used a uniform grid acceleration data structure to store triangular geometry. The grid transversal kernel searches this grid using a 3D-DDA algorithm. The transverser kernel is implemented as multipass approach, looping over all grid cells along a given path. The ray-triangle intersector performs the ray-triangle intersection test over a stream of ray-voxels, returning a stream of ray-triangle intersections. The intersector kernel implementation is similar to the implementation proposed by Carr [109]. Finally, the shader kernel computes the color resulting from ray-surface scattering.

Similar ray tracing implementations have been developed by others. These implementations use the same set of kernels, differing mostly on some specific kernel implementations. Christen [110] described an implementation where the scene geometry is stored in two dimensional textures, and used early Z-culling on the transverser and intersector kernel. Kerlsson [111] developed a similar ray tracing implementation on the GPU, but using a transversal algorithm based on proximity clouds [112]. Also his ray-triangle intersection test was based on the algorithm proposed by Moller and Trumbore [113].

3.2.3 Photon Mapping

Photon tracing is a natural extension of ray tracing and therefore can be implemented on GPU using a similar procedure. Purcell et al. [34, 108] has extended the ray tracing streaming abstraction in order to model the photon mapping method as a stream computation. This approach performs a breadth-first stochastic photon tracing and uses a grid-based map to store photons. Also, an alternative grid representation is proposed to approximate the near-neighbour density estimation. This grid-based photon map allows for faster storage of photons using a combination of stencil buffers and vertex programs. This approach allows for the construction of the data structure for the storage of photons entirely on GPU.

A different approach has been proposed by Lavignotte and Paulin [32, 114] to simulate global illumination effects using photon mapping. This approach is based on the *photon splatting method* [115], where the radiance estimator is evaluated by splatting the contribution of photons onto the image plane. Lavignotte and Paulin [116] have also presented a more accurate method for photon density estimation which accounts for density functions with bounded support. In order to achieve an adaptive image reconstruction, the proposed density estimator also uses a composition of photon splatting images, each using kernels with different support sizes. This adaptiveness provide more accurate reconstruction of sharp discontinuities, like caustics and shadows.

Larsen [33] has proposed a photon tracing approach for computing a global illumination solution where each global illumination effect is computed separately. The illumi-

nation components considered in this approach are direct, specular, caustic and indirect illumination. Photon hits are distributed and clustered according to a variation of the selective photon tracing method [17].

The indirect illumination component is stored on a light map and changes are incrementally added to or subtracted from it. Caustic photons are computed on the CPU and stored on a linear structure. Since caustic photons are usually localized, the photon caustic image is generated by projecting them over the image plane, counting and filtering the results. The specular and direct illumination components are gathered using traditional environment map and soft shadow techniques [117], respectively. Larsen also proposed a technique to compute the indirect illumination component more efficiently by organizing the photons responsible for indirect illumination according to the scene topology [118].

3.2.4 Texture Projection

The techniques described previously mainly deal with implementations of known light transport algorithms using current graphics hardware. However, some techniques have been developed that are not based exclusively on a particular underlying light transport method. Most of these techniques are based on texture projection schemes, using graphics card textures for caching the illumination samples and texture units to perform the interpolation and filtering operations.

Corrective textures [119] is a technique that augments an interactive rendering solution using a number of textures obtained from a high-quality global illumination solution.

The interactive solution visualizes all geometric features of the scene and is obtained using standard local illumination rendering. Each object or group of objects has a set of corrective textures associated to it. These textures are positioned around the associated object and describe the difference between the interactive solution and the high-quality global illumination solution as seen from the corrective texture center of projection. During the rendering pass, the corrective textures associated with a given object are projected toward it using a projective texture scheme.

Bastos et al. [120, 121] proposed a similar technique for visualization of scenes with diffuse and non-diffuse illumination components. The technique separates the view dependent and view independent components and combines them during rendering. The diffuse component is computed using a traditional radiosity solver and rendered using standard graphics hardware rendering. The view independent component is decomposed into irradiance and reflectance textures that are composed using convolution operators during the rendering stage.

A less conventional use of textures is explored by the *global ray-bundle tracing* method proposed by Szirmay-Kalos [29, 30]. The global ray-bundle technique performs the transport of radiance through a set of parallel rays simultaneously using textures and orthographic rendering. Instead of using single rays or paths to transport radiance, global ray-bundle exploits radiance directional coherence and transport, on a single pass, the radiance information along all rays that pass through a given *transillumination plane* [122].

In this method, the radiance transport is accomplished by initially defining a plane

inside the environment. This plane is then used to transfer radiance samples along parallel rays from a set of emitter patches toward a set of receiver patches. Radiance is transported along the transillumination plane by rendering both emitter and receiver patches onto the transillumination plane using orthographic projection. Emitters and receivers are rendered onto separated buffers, one for each side of the plane. In order to determine the amount of radiance that is transferred from emitters to receivers, both buffers are scanned and the radiance leaving a emitter towards a receiver through a given pixel is computed and accumulated on the receiver patch. This technique uses the z-buffer capabilities of the graphics card to determine the visibility between a set of emitters and a set of receivers.

A related technique proposed by Heidrich et al. [123] uses precomputed visibility information stored in object space to efficiently account for self-shadowing and local light scattering for BRDF evaluation. Their approach uses a variation of horizon maps together with dependent texture lookups to compute the multiple local bounces of light at a given point. This technique was extended by Daubert et al. [124] to compute a global illumination solution on general geometries.

3.3 Summary

This chapter has presented the relevant work developed recently involving interactive rendering and graphics accelerated methods for global illumination. We have described common techniques for interactive rendering of global illumination, based on caching schemes and incremental updates of the final solution. We also described techniques that

make use of current graphics hardware features to perform the visualization as well as to accelerate the transfer of radiance over the environment.

In the following chapters we present pencil light transport, which can be seen as a generalization of the transillumination method presented by Szirmay-Kalos [29, 30] and the quasi-Monte Carlo schemes explored by Heidrich et al. [123] and Daubert et al. [124].

Chapter 4

Pencil Object

Pencil light transport is a method that performs the transfer of radiance through a set of points positioned in free space. Each of these points can be used to define an abstraction called a *pencil*. This chapter describes the general idea behind this abstraction, as well as details of its internal data structure and its required operations.

4.1 Pencil Geometry

The term *pencil* has been used in computer graphics to specify the flow of energy along a colection of rays [54, 125, 126]. An important reference of the use of pencils to describe a method for light transport is pencil tracing [125]. Pencil tracing makes use of paraxial approximation theory to create pencils of rays that can be efficiently tested for intersection with the environment. Also, the maximum spread angle of a pencil of paraxial rays can

be computed based on a given tolerance error. Pencil objects, however, are not directly related to these techniques. The idea of a pencil object is centered on the projective geometry definition of a pencil.

A *pencil* is defined in projective geometry as a set of lines that pass through a common point [127]. This unique point where all lines in the set intersect is called the *center of projection* (COP). Figure 4.1 shows the basic geometry of a pencil and its center of projection.

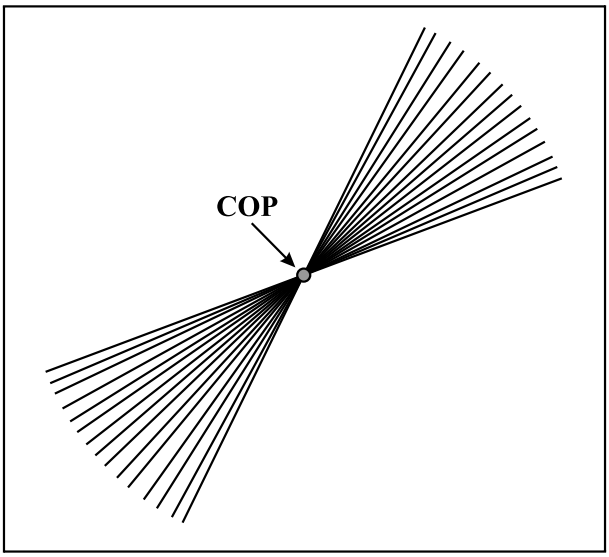
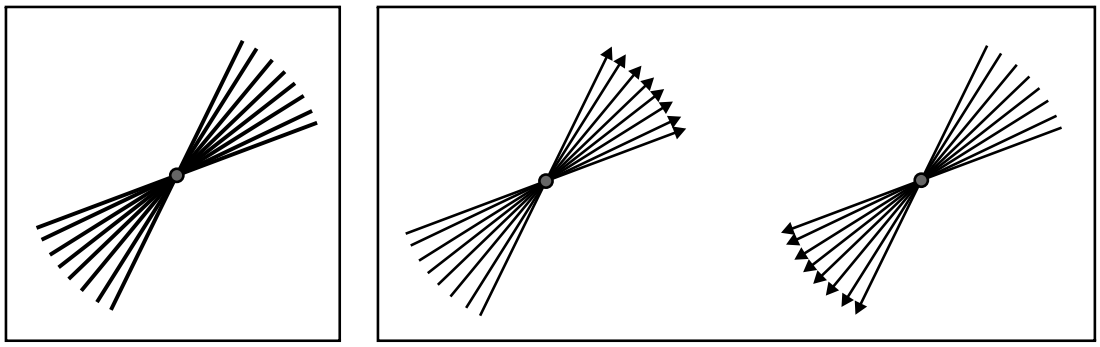


Figure 4.1: Geometry of a pencil. A pencil is a set of lines the pass through a center of projection (COP.)

This standard definition of a pencil is based on a collection of unoriented lines. However, since our intention is to use the pencil abstraction as a mechanism to transport radiance along the scene, unoriented lines are not sufficient. This is due to the fact that the radiance being transferred along a direction $\hat{\omega}$ may not be the same as the radiance

being transferred in the opposite direction $-\hat{\omega}$.

A better primitive for light transport is a oriented line, or simply a ray. Unlike a unoriented line, a ray specifies a direction of flow, allowing us to make a distinction between lines passing through a point $\mathbf{x}$ with directions $\hat{\omega}$ and $-\hat{\omega}$. We define a *pencil of rays* as a set of infinite rays that pass through a common point. Figure 4.2 shows the ambiguity problem caused by a pencil of lines and how a pencil of rays can be used to address this issue.



(a) (b)

Figure 4.2: Pencils of lines and pencils of rays. (a) *Pencils of lines* do not describe the direction of flow. (b) *Pencils of rays* can be used to specify the two possible directions of flow.

In order to fully specify a pencil of rays, we need to define its coverage in addition to its center of projection. The coverage of a pencil is given by the set of directions it encloses. Letting $R(\mathbf{x}, \hat{\omega})$ be the ray that passes through $\mathbf{x}$ with direction given by $\hat{\omega}$, a pencil of rays $P(\mathbf{c}, \Omega)$ with center of projection $\mathbf{c}$ and coverage given by the set of

directions Ω can be formally defined as

$$P(\mathbf{c}, \Omega) = \{\forall \hat{\omega} \in \Omega \mid R(\mathbf{c}, \hat{\omega})\}. \tag{4.1}$$

When developing an algorithm for light transport it is desirable to transfer as much information as possible for a given computational budget. Having this idea in mind, it becomes useful to construct pencils of rays that cover all possible directions around a given center of projection. We define an *omni-pencil of rays* as a pencil of rays which coverage Ω is given by the complete sphere of directions. Figure 4.3 illustrates the geometry of an omni-pencil of rays. Most pencils used hereafter are omni-pencils of rays, so pencils will be considered as *omni-pencils of rays* unless stated otherwise. Note that in an omni-pencil of rays we have *two* rays for each line through the center of projection.

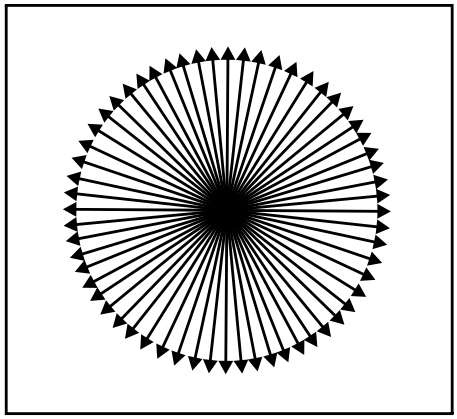


Figure 4.3: Geometry of an omni-pencil of rays. Omni-pencil of rays has coverage given by the complete set of directions.

Our intention is to define an abstraction that can be used on light transport algo-

rithms. A ray is a general primitive that is commonly used for light transport. However, light transport algorithms simulate the transfer of radiance from one surface point to another, which can ultimately be represented by a ray *segment*. In order to be able to represent ray segments, a pencil object must also be able to associate depth information with all ray endpoints. Figure 4.4 shows a pencil of rays with source and target depths attached to each ray. Attaching depth information to each ray of a pencil allows us to represent a set of ray segments passing through a common point.

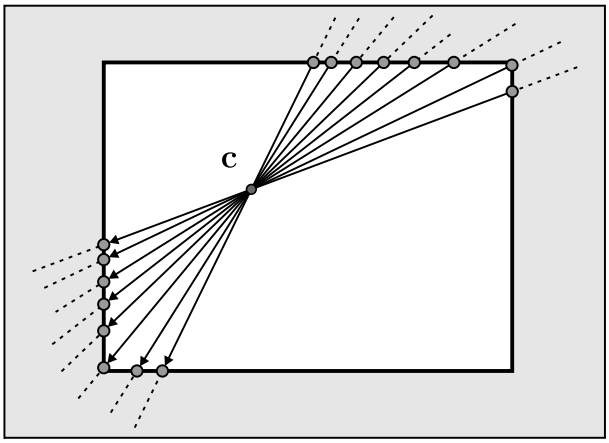


Figure 4.4: Attaching depth information to each ray enables the construction of pencils of ray segments.

4.2 Pencil Data

A *pencil object* can be defined basically as a *center of projection* together with some associated *directional data*. The main attribute of a pencil object is its center of projection. The directional data associated to a pencil is defined relative to its center of projection

and a set of directions.

Since we are using omni-directional pencils of rays, directional data is represented over the whole sphere of directions. A significant advantage of using omni-pencils of rays is that they are strongly related to environment maps. Like environment maps, omni-pencil of rays are centered at a fixed position and have some information associated to each direction at their center of projection. This means that environment maps can be used to represent and store directional data associated with a center of projection.

This relationship between pencils and environment maps allows us to efficiently represent and store the directional data of a pencil on graphics hardware using a number of environment maps. Storing the directional data on separate environment maps offers some advantages, since we can use a priori knowledge about the information being stored and assign different resolutions for each type of data. For instance, smooth directional functions can be stored using environment maps with lower resolution. Also, some information may stay constant over the simulation process, such as distance along ray segments, and therefore can be stored in separate environment maps as well. The proposed pencil light transport algorithm associates *exitant radiance*, a *directional distance* and a *directional pencil density function* with the directional data of a pencil.

The exitant radiance is stored as a two-dimensional texture at pencil position in order to provide intermediary storage during a light transport simulation. Considering that radiance is transferred only between visible surface points during a given light bounce, a distance function is used to determine the endpoints of ray segments. Since the distance

function specifies the distance between the center of projection of a given pencil and the closest surface point in the direction $\hat{\omega}$, the endpoints of a ray segment passing through the center of projection and direction of flow $\hat{\omega}$ are determined using the distance entries at directions $-\hat{\omega}$ and $\hat{\omega}$. We also define a directional pencil density function at pencil. This function is used to correct for the non-uniform distribution of pencils over the scene, and is described in more detail in the next chapter.

4.3 Pencil Operations

The pencil object abstraction used by the pencil light transport is capable of performing two basic operations using the center of projection and the directional data associated to it. These two operations are *gathering* and *projection*. Since a pencil's directional data can be represented in a similar way as environment maps, both pencil operations have a close relation to operations performed by environment mapping techniques.

A pencil gather operation consists of acquiring the corresponding information from a given direction with respect to the pencil's center of projection. This operation can be implemented as a pinhole camera rendering of the scene to the environment map.

A pencil projection operation of some information (such as radiance) onto the environment consists of determining for each surface point being processed the corresponding value arriving from the pencil's center of projection. This operation can be implemented using a projective texture lookup into an environment map. On a graphics hardware implementation, this texture lookup is performed while running a fragment program. That

means a pencil projection is performed implicitly while performing another task, like a pencil gathering operation or while running another fragment program.

Computationally efficient pencil operations are important for an optimized pencil transport implementation. Since these operation are closely related to environment mapping operations, the correct choice of an environment map scheme is essential to the overall performance of an implementation. As will be shown in the next chapter, a highly desirable property of a environment map scheme for a pencil light transport algorithm is the ability to handle a large number of environment map lookups per rendering pass. Another desirable property is efficient use of the available texture memory, so no radiance samples are wasted during a pencil projection operation. Even though these properties do not improve the complexity order of the algorithm, they may have a significant impact on the relative overall performance of a pencil transport implementation. We have studied some alternative environment map representations, but it should be emphasized that the pencl light transport algorithm itself is independent of the particular environment mapping scheme used.

Parabolic environment maps can represent the radiance information from one hemisphere using a standard two-dimensional texture and have a reasonably efficient texture lookup formula. However, the parabolic projection is non-linear, mapping line segments from world space to arcs in texture space. This make the environment map acquisition pass unreasonably expensive for an arbitrarily positioned center of projection, requiring the scene to be finely tessellated. Another problem is that information from an hemi-

sphere is projected onto a circle, using only approximately 78.5% of the available texture memory.

Cube environment maps can be used to represent the complete radiance information around a point using six two-dimensional textures. The cube projection maps straight lines in three-space to straight lines on each face, requiring no geometry processing for the environment map acquisition. This allows for the positioning of centers of projection arbitrarily close to surfaces. However, cubemaps currently have a implementation with limited functionality on current graphics hardware, and many features that are available to two-dimensional textures, such as floating-point interpolation, are missing. Also, hardware cubemap setup organization requires the use of separate rendering contexts to capture the information for each face, making the acquisition process of an cubemap slower than acquiring six faces on a single context. Yet, the main issue limiting the use of cube environment maps in a pencil transport implementation is the difficulty of combining several cubemaps into a bigger cubemap. This is a severe problem, since GPUs have a relatively small number of texture units. Even though some of these issues are likely to be addressed in the future, with the introduction of frame buffer objects (FBO) or with the availability of floating-point support for cubemaps, these limitations led us to exclude the use of the internally implemented cubemap as the environment map of choice in our current implementation of the pencil operations. Cubemaps can also be implemented using shader programs and stored in rectangular textures. However, this approach does not exploit the available hardware support for filtering and interpolation.

We have developed a new environment map projection scheme that is more suitable for pencil transport algorithms than parabolic or cube maps on the graphics hardware available to us at the time of implementation of our prototype. Our new environment map scheme projects the radiance information from the whole sphere of directions onto one single two dimensional texture, having an octahedron as the base primitive for projection.

4.3.1 Pyramid and Octahedral Environment Mapping

We start our formulation of the octahedral projection by generalizing the parabolic projection to use superquadric paraboloids as base primitives for projection. A superquadric paraboloid has the general form

$$\frac{z}{c} = \left|\frac{x}{a}\right|^n + \left|\frac{y}{b}\right|^n , \tag{4.2}$$

and defines a generalized form of the paraboloid, which is the base primitive for the parabolic projection. In order to accomplish this generalization, initially consider the parabolic texture lookup formulas

$$u = \frac{\hat{\mathbf{v}}_x}{\hat{\mathbf{v}}_z + 1} \qquad \text{and} \qquad v = \frac{\hat{\mathbf{v}}_y}{\hat{\mathbf{v}}_z + 1} , \tag{4.3}$$

where $\hat{\mathbf{v}}$ is the unit vector from center of projection $\mathbf{c}$ to a point $\mathbf{x}$ in the forward hemisphere [88, 89].

A strong requirement in equation 4.3 is that vector $\hat{\mathbf{v}}$ must be normalized. However,

this requirement can be removed using the lookup formulas

$$u = \frac{\vec{\mathbf{v}}_x}{\vec{\mathbf{v}}_z + d} \qquad \text{and} \qquad v = \frac{\vec{\mathbf{v}}_y}{\vec{\mathbf{v}}_z + d}\,, \tag{4.4}$$

where $d = |\vec{\mathbf{v}}| = \sqrt{\vec{\mathbf{x}}^2 + \vec{\mathbf{y}}^2 + \vec{\mathbf{z}}^2}$. These lookup formulas perform the normalization of $\vec{\mathbf{v}}$ implicitly. Both formulas have nearly the same computational cost, but these simple modifications lead to some important observations.

First, parabolic projection uses an Euclidean norm $d = |\vec{\mathbf{v}}|_2$ to compute the length of vector $\vec{\mathbf{v}}$. However, changing the metric norm used to compute the length of $\vec{\mathbf{v}}$ allows us to define projections using different base primitives. It turns out that using the metric norm L_p to compute the vector length

$$d = |\vec{\mathbf{v}}|_p = \sqrt[p]{|\vec{\mathbf{v}}_x|^p + |\vec{\mathbf{v}}_y|^p + |\vec{\mathbf{v}}_z|^p} \tag{4.5}$$

defines a projection that uses a superquadric paraboloid with exponent p as base primitive. Some norms that deserve special attention are L_1, L_2 and L_∞ [128], respectively given by

$$|\vec{\mathbf{v}}|_1 = |\vec{\mathbf{v}}_x| + |\vec{\mathbf{v}}_y| + |\vec{\mathbf{v}}_z| \quad , \tag{4.6}$$

$$|\vec{\mathbf{v}}|_2 = \sqrt{\vec{\mathbf{v}}_x^2 + \vec{\mathbf{v}}_y^2 + \vec{\mathbf{v}}_z^2} \qquad \text{and} \tag{4.7}$$

$$|\vec{\mathbf{v}}|_\infty = \max(|\vec{\mathbf{v}}_x|, |\vec{\mathbf{v}}_y|, |\vec{\mathbf{v}}_z|) \quad . \tag{4.8}$$

A superquadric paraboloid with exponent p, as defined in equation 4.5, has a super-elliptic cross section with exponent p along the z-axis. Both cases with $p = 1$ and $p = \infty$ define pyramid frustums as base primitive, although with different orientations. When $p = \infty$, the pyramid frustum is aligned to x and y axis, while for $p = 1$ the pyramid frustum is aligned to diagonals $x = y$ and $x = -y$. However, both pyramids' bases are inscribed in the same texture square. Figure 4.5 shows the base primitives for superquadric paraboloids with p equal to 1, 2 and ∞.

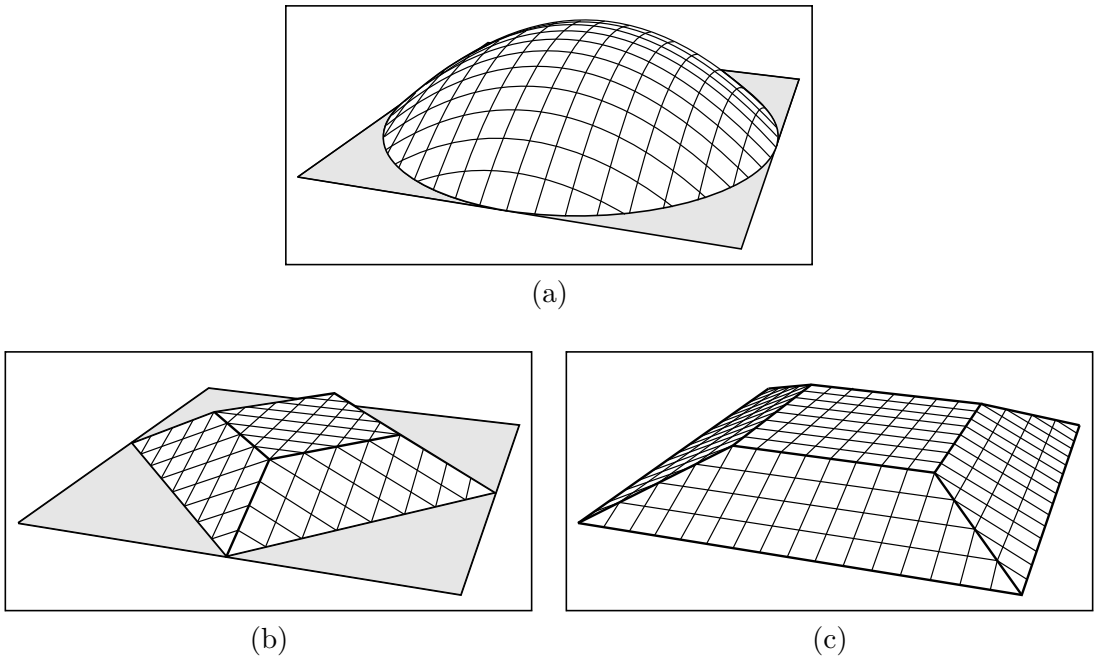


Figure 4.5: Superquadric paraboloids used as projection bases: (a) using $p = 2$ (paraboloid), (b) using $p = 1$, (c) using $p = \infty$.

A second observation is that by removing the $\vec{\mathbf{v}}_z$ component from the norm, the base primitive in the projection is changed to a superquadric cone. Based on norms L_1, L_2

and L_∞ defined respectively in equations 4.6 to 4.8, the norms

$$|\vec{\mathbf{v}}|'_1 = |\vec{\mathbf{v}}_x| + |\vec{\mathbf{v}}_y| \quad , \tag{4.9}$$

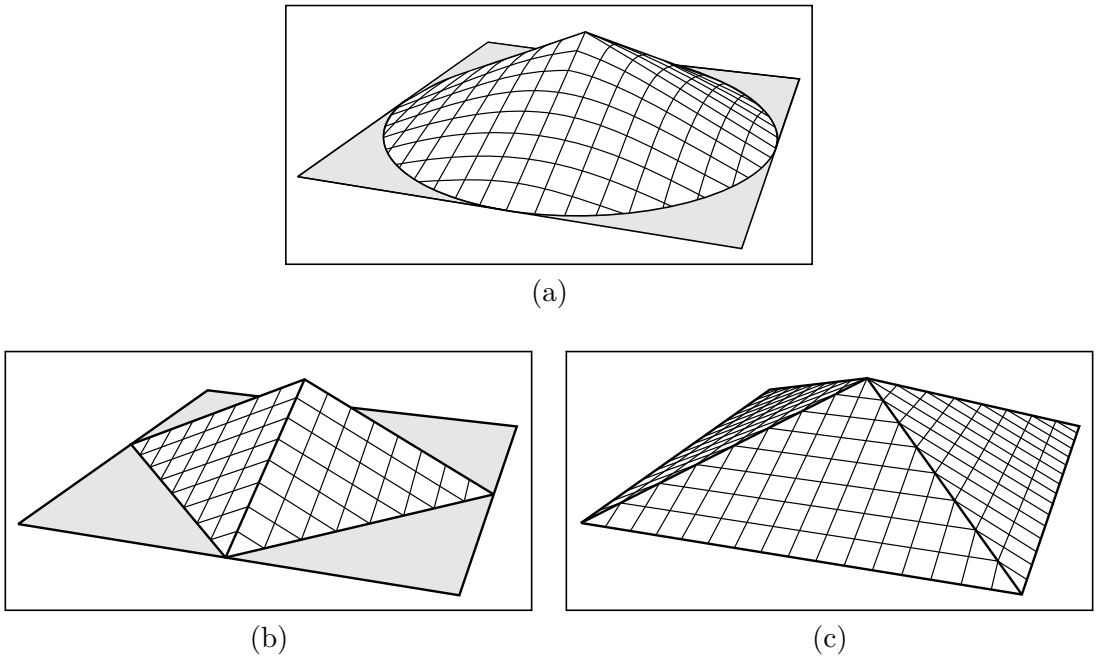
$$|\vec{\mathbf{v}}|'_2 = \sqrt{\vec{\mathbf{v}}_x^2 + \vec{\mathbf{v}}_y^2} \quad \text{and} \tag{4.10}$$

$$|\vec{\mathbf{v}}|'_\infty = \max(|\vec{\mathbf{v}}_x|, |\vec{\mathbf{v}}_y|) \tag{4.11}$$

define projection schemes that use a cone as a base primitive for the L'_2 norm, as well as pyramids with square bases for the L'_1 and L'_∞ norms. The pyramids defined by the L'_1 and L'_∞ norms have the same orientation as the pyramid frustum described for the superquadric paraboloids. Figure 4.6 shows the base primitives for superquadric cones when p is 1, 2 and ∞.

The metric norms L'_1 and L'_∞ are especially interesting because they have a *partial linear* property. This is a important property because it allows for a projection to be described as a combination of linear projections, one for each face of the pyramid. Also, the lookup formulas for both projection schemes can be efficiently implemented in current graphics hardware. Furthermore, each projection can be optimized differently, allowing even better performance.

Consider the projection using $d = |\vec{\mathbf{v}}|'_\infty = \max(|\vec{\mathbf{v}}_x|, |\vec{\mathbf{v}}_y|)$. This projection maps a hemisphere onto an axis aligned square, as in figure 4.6(c). We will call this projection a



(a) (b) (c)

Figure 4.6: Superquadric cones used as projection bases: (a) using $p = 2$ (cone), (b) using $p = 1$, (c) using $p = \infty$.

pyramid projection, and its texture lookup formulas are given by

$$u = \frac{\vec{\mathbf{v}}_x}{\vec{\mathbf{v}}_z + \max(|\vec{\mathbf{v}}_x|, |\vec{\mathbf{v}}_y|)} \qquad \text{and} \qquad v = \frac{\vec{\mathbf{v}}_y}{\vec{\mathbf{v}}_z + \max(|\vec{\mathbf{v}}_x|, |\vec{\mathbf{v}}_y|)}\,, \tag{4.12}$$

where $\vec{\mathbf{v}}$ is not required to be an unit vector. The geometry of the pyramid projection is detailed in figure 4.7.

The pyramid projection has clear advantages for an implementation based on graphics hardware. Since most instructions operate on vector variables (except the division), it enables the computation of four texture lookups using almost the same number of instructions required for one texture lookup alone. Listing 4.1 shows some simple code to

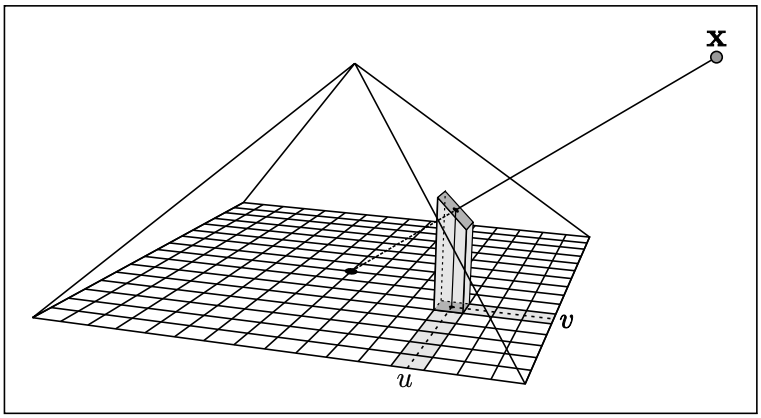


Figure 4.7: Geometry of the pyramid projection. A pyramid is used as base primitive to find the texture coordinates (u, v).

compute the corresponding texture coordinates of a point $\mathbf{p}$ with respect to four arbitrary centers of projection $\mathbf{c}_i$, with the same orientation. Since all centers of projection have the same orientation, it is possible to optimize the computation of the four vectors from $\mathbf{c}_i$ to $\mathbf{x}$ in the corresponding coordinate system. Considering that the matrix T defines the orientation of the coordinate system of centers of projection $\mathbf{c}_i$, vectors $\vec{\mathbf{v}}_i$ can be computed using

$$\vec{\mathbf{v}}_i = T(\mathbf{p} - \mathbf{c}_i) = T\mathbf{p} - T\mathbf{c}_i \,, \tag{4.13}$$

where $T\mathbf{p}$ can be computed in the vertex program and linearly interpolated, and $T\mathbf{c}_i$ only needs to be computed once in a pre-processing step. This optimization significantly reduces the load of the fragment program, but assumes that centers of projection have the same orientation. Also, centers of projection are organized having one coordinate per register in order to fully use the GPU's parallel computation capabilities. This code com-

Listing 4.1 Texture lookup code for pyramid projection.

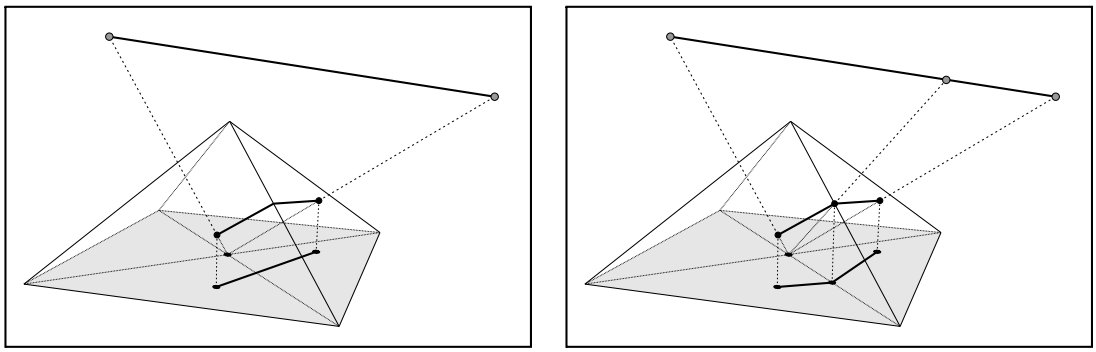
```
// f[TEX0] =  [ p'_x , p'_y , p'_z , · ] = Tp
// p[0]    =  [ c'_1x , c'_2x , c'_3x , c'_4x ] = (Tc)_x
// p[1]    =  [ c'_1y , c'_2y , c'_3y , c'_4y ] = (Tc)_y
// p[2]    =  [ c'_1z , c'_2z , c'_3z , c'_4z ] = (Tc)_z

SUB R1, f[TEX0].x, p[0];    // R1 = [v1_x , v2_x , v3_x , v4_x ] = V_x
SUB R2, f[TEX0].y, p[1];    // R2 = [v1_y , v2_y , v3_y , v4_y ] = V_y
SUB R3, f[TEX0].z, p[2];    // R3 = [v1_z , v2_z , v3_z , v4_z ] = V_z

MAX R4, |R1| , |R2|;
ADD R4, R4 , R3;        // R4 =  V_z + max(|V_x| , |V_y|) = V_w

RCP R4.x, R4.x;
RCP R4.y, R4.y;
RCP R4.z, R4.z;
RCP R4.w, R4.w;         // R4 =  1/V_w

MUL R5.xz, R1.xxyy, R4.xxyy;
MUL R5.yw, R2.xxyy, R4.xxyy;    // R5 =  [ u1 , v1 , u2 , v2 ]
MUL R6.xz, R1.zzww, R4.zzww;
MUL R6.yw, R2.zzww, R4.zzww;    // R6 =  [ u3 , v3 , u4 , v4 ]
```

putes four texture coordinates using 13 GPU assembly instructions, or 3.25 instructions per projection on average.

A texture lookup can be used to directly implement the projection operator for one hemisphere. On the other hand, the gathering operator requires the rendering of the scene geometry of a single hemisphere to a texture. The rendering must be done so that the acquired information conforms to a pyramid projection and can be used on further pyramid projection lookups.

Regarding the acquisition step, pyramid projection can benefit from the fact that each

face of the pyramid is defined by a linear projection. A linear projection maps straight lines from world space to straight lines in texture space, so the projection is performed correctly as long as the projected geometry lies entirely on a pyramid's face. However, when an object is projected onto more than one face, artifacts can happen due to the use of different projections for each face. Figure 4.8 illustrates the distortion artifacts that can happen when a line is projected on two adjacent faces.



(a) (b)

Figure 4.8: Interpolation across faces using pyramid projection: (a) Linear interpolation is not correct across faces. (b) Geometry must be rendered separately for each face.

One strategy to avoid the transition artifacts is to split the geometry along the planes that pass through the pyramid edges. This splitting guarantees that each primitive projects onto only one pyramid face. Even though this is not a computationally expensive operation, it requires processing of the scene geometry, something that we often want to avoid.

An alternative approach that does not require any processing of the scene geometry is to use a multipass approach. Since rendering using a pyramid projection can be sepa-

rated into four rendering passes each using linear projections, each face can be acquired separately and combined to form the final texture map. Each face can be acquired by rendering the scene geometry using its linear projection and filtering out fragments that do not belong to the corresponding face region in texture space. Fragments that do not project through a given pyramid face can be filtered using a simple and efficient test. The linear projection used to render a scene to a pyramid face can be given by the following matrix:

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ q_x & q_y & h & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.14}$$

where q_x and q_y define the pyramid face being used as well as the corresponding quadrant in texture space. The hemisphere being rendered is specified by h. Table 4.3.1 shows the parameters q_x, q_y and h corresponding to each pyramid face illustrated in figure 4.9, as well as the required condition for accepting a fragment. Even though this approach requires the scene geometry to be rendered four times, these four passes can be implemented efficiently, not causing a significant increase in the processing time when compared to a single pass rendering. Shader program implementation of cubemaps can acquire the full sphere of directions using six passes, but have a more costly texture lookup, and interpolation and MIPmap filtering are more complex. In other words, cube maps are slightly cheaper to render into, but more expensive to lookup from.

Region	Condition	q_x	q_y
A	$u \geq v$	1	0
B	$u \leq -v$	0	-1
C	$-u \geq v$	-1	0
D	$u \leq v$	0	1

Hemisphere	Condition	h
Forward	$\vec{\mathbf{v}}_z \geq 0$	1
Backward	$\vec{\mathbf{v}}_z \leq 0$	-1

Table 4.1: Parameters used to specify the linear projection of each pyramid face.

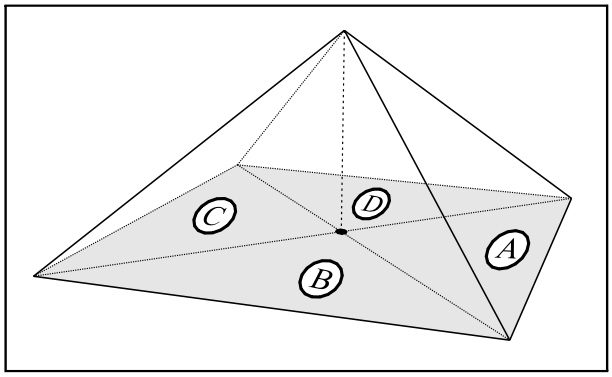


Figure 4.9: Regions corresponding to each face on a pyramid projection.

Pyramid projections are suitable for handling hemispherical environment maps or for projecting hemispherical shadow maps. Two pyramid maps can be used to capture the radiance over the whole sphere of directions. However, this approach requires an additional computational cost for selecting the correct texture. Also, symmetry is a desirable property for an environment map. Having two hemispheres on two separate textures reduces the ability for the environment map to use filtering schemes supported on graphics hardware, such as MIP-mapping [41].

Observe that allowing $d = |\vec{\mathbf{v}}|_1$ defines a projection that maps a hemisphere onto a *diagonally* aligned square region that covers half the texture area, as in figure 4.6(b). This projection is a candidate for representing a complete environment map on a single two-

Region	Condition	q_x	q_y
A	$u \geq 0, v \geq 0$	1	1
B	$u \geq 0, v \leq 0$	1	-1
C	$u \leq 0, v \leq 0$	-1	-1
D	$u \leq 0, v \geq 0$	-1	1

Table 4.2: Parameters for each face of an octahedral projection.

dimensional texture. In fact, it turns out that it is possible to combine the projections from forward and backward hemispheres seamlessly on complementary regions of the texture, and still use an efficient texture lookup. We call this environment map projection an *octahedral projection*.

Like the pyramid projection, the octahedral projection can be separated into multiple linear projections, one for each face. The projection matrix to render to each of these faces can be obtained using the same projection matrix specified in (4.14), but using the parameters q_x and q_y from table 4.3.1. The regions corresponding to each face are shown in figure 4.10.

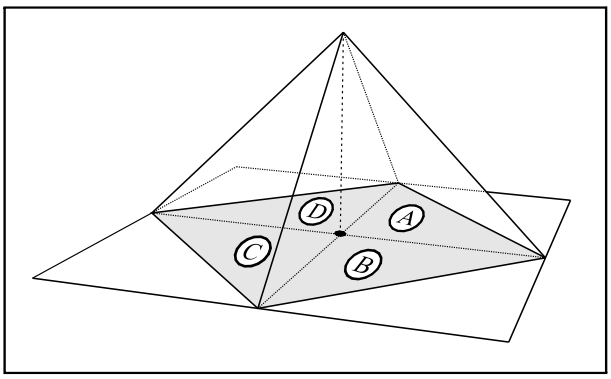


Figure 4.10: Regions corresponding to each face on an octahedral projection.

Although equation 4.16 maps an hemisphere to a square which covers half the texture

area, both forward and backward hemispheres project onto the same region. Flipping one square to the outer regions enables the storage of the information from both hemispheres into one single square region. This can be done using the flipping matrix given by

$$F = \begin{bmatrix} 0 & -S_x \cdot S_y & 0 & S_x \\ -S_x \cdot S_y & 0 & 0 & S_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.15}$$

where S_x and S_y are the signs of $\vec{\mathbf{v}}_x$ (or u) and $\vec{\mathbf{v}}_y$ (or v), respectively. The flipping matrix can pre-multiply the projection matrix in order to map the information from an hemisphere to the outer regions of the texture, as illustrated in figure 4.11. A useful property of the flipping matrix is that it allows a continuous transition between forward and backward hemispheres.

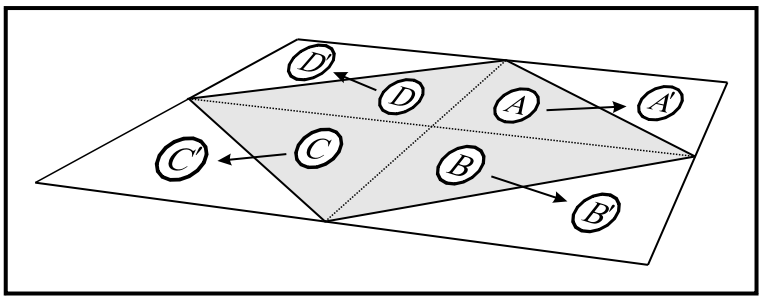


Figure 4.11: Reflected regions on an octahedral projection.

Since the lookup coordinates for one hemisphere of the octahedral projection are given

by

$$u = \frac{\vec{\mathbf{v}}_x}{\vec{\mathbf{v}}_z + |\vec{\mathbf{v}}_x| + |\vec{\mathbf{v}}_y|} \qquad \text{and} \qquad v = \frac{\vec{\mathbf{v}}_y}{\vec{\mathbf{v}}_z + |\vec{\mathbf{v}}_x| + |\vec{\mathbf{v}}_y|}\,. \tag{4.16}$$

and the flipping matrix can be expressed as

$$u' = -S_x \cdot S_y \cdot v + S_x \qquad \text{and} \qquad v' = -S_x \cdot S_y \cdot u + S_y\,, \tag{4.17}$$

the lookup coordinates for the octahedral projection can be combined on a single equation using the sign of $\vec{\mathbf{v}}_z$:

$$u = \left(\frac{-S_x \cdot S_y \cdot \vec{\mathbf{v}}_y}{W} + S_x\right)\alpha + \left(\frac{\vec{\mathbf{v}}_x}{W}\right)(1-\alpha) \tag{4.18}$$

$$v = \left(\frac{-S_x \cdot S_y \cdot \vec{\mathbf{v}}_x}{W} + S_y\right)\alpha + \left(\frac{\vec{\mathbf{v}}_y}{W}\right)(1-\alpha) \tag{4.19}$$

where $\alpha = (\vec{\mathbf{v}}_z < 0)$ and $W = |\vec{\mathbf{v}}_x| + |\vec{\mathbf{v}}_z| + |\vec{\mathbf{v}}_z|$. The factor W is derived from the fact that $h \cdot \vec{\mathbf{v}}_z = |\vec{\mathbf{v}}_z|$ for both forward and backward hemispheres.

Equation 4.18 can be expanded to

$$u = \frac{\left(-S_x \cdot S_y \cdot \vec{\mathbf{v}}_y + S_x|\vec{\mathbf{v}}_x| + S_x|\vec{\mathbf{v}}_y| + S_x|\vec{\mathbf{v}}_z|\right)\alpha + (\vec{\mathbf{v}}_x - \alpha\vec{\mathbf{v}}_x)}{W}. \tag{4.20}$$

Using $S_y\vec{\mathbf{v}}_y = |\vec{\mathbf{v}}_y|$ and $S_x|\vec{\mathbf{v}}_x| = \vec{\mathbf{v}}_x$ allows equation 4.20 to be simplified to

$$u = \frac{(S_x|\vec{\mathbf{v}}_x| + S_x|\vec{\mathbf{v}}_z|)\,\alpha + (\vec{\mathbf{v}}_x - \alpha\vec{\mathbf{v}}_x)}{W} \tag{4.21}$$

$$u = \frac{\alpha S_x|\vec{\mathbf{v}}_z| + \vec{\mathbf{v}}_x}{W} \tag{4.22}$$

$$u = \frac{\vec{\mathbf{v}}_x + S_x|\vec{\mathbf{v}}_z|(\vec{\mathbf{v}}_z < 0)}{W} \tag{4.23}$$

where $(\vec{\mathbf{v}}_z < 0)$ equals 1 if the condition is met and 0 otherwise. Following the same procedure on equation 4.19 results in

$$v = \frac{\vec{\mathbf{v}}_y + S_y|\vec{\mathbf{v}}_z|(\vec{\mathbf{v}}_z < 0)}{W} \tag{4.24}$$

Finally, observing that $|\vec{\mathbf{v}}_z|(\vec{\mathbf{v}}_z < 0) = \max(-\vec{\mathbf{v}}_z, 0)$ leads to the final equations for the octahedral texture lookup:

$$u = \frac{\vec{\mathbf{v}}_x + S_x \max(-\vec{\mathbf{v}}_z, 0)}{W} \qquad \text{and} \qquad v = \frac{\vec{\mathbf{v}}_y + S_y \max(-\vec{\mathbf{v}}_z, 0)}{W}\,. \tag{4.25}$$

The octahedral texture lookup can be efficiently implemented on current graphics hardware. Projecting four points requires only five extra instructions over that required for projecting four points using the pyramidal projection. Listing 4.2 shows the code to compute the corresponding octahedral projection texture lookup coordinates of a point $\mathbf{p}$ relative to four arbitrary centers of projection $\mathbf{c}_i$ with the same orientation. The four texture coordinates are obtained using 18 instructions, or 4.5 instructions per projec-

Listing 4.2 Texture lookup code for octahedral projection.

```
// f[TEX0] =  [ p'_x , p'_y , p'_z , · ] = Tp
// p[0]    =  [ c'_1x , c'_2x , c'_3x , c'_4x ] = (Tc)_x
// p[1]    =  [ c'_1y , c'_2y , c'_3y , c'_4y ] = (Tc)_y
// p[2]    =  [ c'_1z , c'_2z , c'_3z , c'_4z ] = (Tc)_z

SUB R1, f[TEX0].x, p[0];    // R1 = [v_1x , v_2x , v_3x , v_4x ] = V_x
SUB R2, f[TEX0].y, p[1];    // R2 = [v_1y , v_2y , v_3y , v_4y ] = V_y
SUB R3, f[TEX0].z, p[2];    // R3 = [v_1z , v_2z , v_3z , v_4z ] = V_z

ADD R4, |R1|, |R2|;
ADD R4,  R4 , |R3|;    // R4 = |V_z| + |V_x| + |V_y| = V_w

RCP R4.x, R4.x;
RCP R4.y, R4.y;
RCP R4.z, R4.z;
RCP R4.w, R4.w;        // R4 = 1/V_w

MAX R7, -R3, 0;        // R7 = max(-V_z, 0)
SGN R8, R1;            // R8 = max(V_x)
MAD R1, R1, R8, R7;    // R1 = V_x + sgn(V_x) max(-V_z, 0)

SGN R8, R2;
MAD R2, R2, R8, R7;    // R2 = V_y + sgn(V_y) max(-V_z, 0)

MUL R5.xz, R1.xxyy, R4.xxyy;
MUL R5.yw, R2.xxyy, R4.xxyy;    // R5 = [ u_1 , v_1 , u_2 , v_2 ]
MUL R6.xz, R1.zzww, R4.zzww;
MUL R6.yw, R2.zzww, R4.zzww;    // R6 = [ u_3 , v_3 , u_4 , v_4 ]
```

tion. This code makes the same assumptions as the code for computing pyramid texture coordinates given in listing 4.1.

The octahedral projection is especially suitable for implementing the pencil transport operator. Besides being computationally efficient, it has other advantages as well. Octahedral projection enables several environment maps to be easily composed in one single two-dimensional texture. This is a significant benefit since the number of texture units available on current graphics hardware is considerably limited. This property extends significantly the number of pencils that can be projected on a single pass. Also, an octahedral environment map has a continuous transition between forward and backward hemispheres and is axially symmetric, making it suitable for MIPmap filtering. Note that replicating the appropriate texels around the boundary gives correct interpolation using only two dimensional hardware interpolation.

4.4 Summary

This chapter presented an abstraction for a pencil object, its internal representation and operations. This abstraction is used in the next chapter as the main mechanism for light transport. We described the three kinds of data associated with directions on pencil objects: exitant radiance, directional distance and directional pencil density. We also described the two main operations required for pencil objects and how they map to a graphics hardware implementation. Finally, we described an efficient environment map projection scheme especially suitable for a pencil light transport implementation.

In the following chapters, we will not depend on the details of the implementation of these objects and their operations. For instance, it is quite possible to implement them with cube maps, and this might make more sense on future hardware with built-in floating-point interpolation and filtering support. Therefore, from now on, we will treat pencil objects as encapsulated entities with certain properties, but will exploit these properties to formulate our light transport algorithm.

Chapter 5

Pencil Transport

The most common formulation of the rendering equation performs integration over hemispheres using incoming radiancies. Several different formulations have been presented over last two decades. Each formulation organizes the rendering integral equation according to different illumination effects that are being supported. Some well known formulations perform integration over surface points or use outgoing radiance. Veach [50] has presented a particularly interesting formulation where the transport operator is separated into a composition of two simpler scattering and propagation operators.

In this chapter we reorganize the scattering-propagation formulation introduced in chapter 2 to arrive at a pencil light transport formulation of the rendering equation. However, we use a slightly different representation. While the Veach formulation makes a implicit distinction between incoming and exitant radiance fields, we make the separation explicit. We extend the scattering-propagation formulation shown in equation 2.33 by

expressing the propagation operator as a composition of a gathering operator and a projection operator. This separation can be done by reparameterizing the propagation operator.

The propagation operator relating the exitant radiance at a surface point $\mathbf{y}$ and the incoming radiance at a surface point $\mathbf{x}$ is defined based on the point $\mathbf{x}$ and the direction $\hat{\omega}$ from $\mathbf{y}$ to $\mathbf{x}$. The propagation operator can be specified in terms of a set of rays connecting visible surfaces. However, it is possible to define the propagation operator using a different parameterization. Pencil light transport defines the propagation operator based on a set of centers of projection distributed on free space instead of on surface points. This modification provides sufficient flexibility to design a light transport method suitable for efficient implementation on current graphics hardware.

A propagation operator can be specified using a set of ray segments, each one of them connecting two visible surface points. In order to derive a formulation of the light transport through centers of projection, consider the propagation operator specified by a set of rays segments that pass through a common point (a center of projection) and with endpoints at the closest surface points along each direction, as illustrated in figure 5.1(a). This set of ray segments can be split into two disjoint sets of ray segments using its center of projection: one set containing ray segments from surface points to the center of projection and another set containing ray segments from the center of projection to surface points, as shown in figures 5.1(b) and (c). Each one of these sets of ray segments can be used to define a separate propagation operator. We will call the *gathering operator* the

operator defined by the set of ray segments from surface points to the center of projection and we will denoted it by $\mathcal{Q}$. We will call the *projection operator* the operator defined by the set of ray segments from the center of projection back onto surface points and will denote it by $\mathcal{R}$. Operators $\mathcal{Q}$ and $\mathcal{R}$ can be formally defined as

$$(\mathcal{Q}g)(\mathbf{c},\hat{\omega}) = g^*(h(\mathbf{c},-\hat{\omega}),\hat{\omega}), \tag{5.1}$$

$$(\mathcal{R}g)(h(\mathbf{c},\hat{\omega}),\hat{\omega}) = g^*(\mathbf{c},\hat{\omega}) \tag{5.2}$$

where $\mathbf{c}$ is the center of projection and $\hat{\omega}$ is the direction of radiance propagation along the center of projection $\mathbf{c}$. Operators $\mathcal{Q} : L \rightarrow L^*$ and $\mathcal{R} : L \rightarrow L^*$ can be interpreted as simplified versions of the propagation operator $\mathcal{G} : L \rightarrow L^*$, so they both operate over an exitant radiance field and produce as result an incident radiance field.

Observing that for a point positioned in free space the incident and exitant radiance fields are equivalent, an equivalence operator can be defined to allow the composition of gathering and propagation operators to reproduce the original propagation operator $\mathcal{G}$. Using the *equivalence operator* $\mathcal{E} : L^* \rightarrow L$ defined by

$$(\mathcal{E}g^*)(\mathbf{x},\hat{\omega}) = g(\mathbf{x},\hat{\omega}). \tag{5.3}$$

allows the creation of the relation

$$\mathcal{G} = \mathcal{R} \circ \mathcal{E} \circ \mathcal{Q}. \tag{5.4}$$

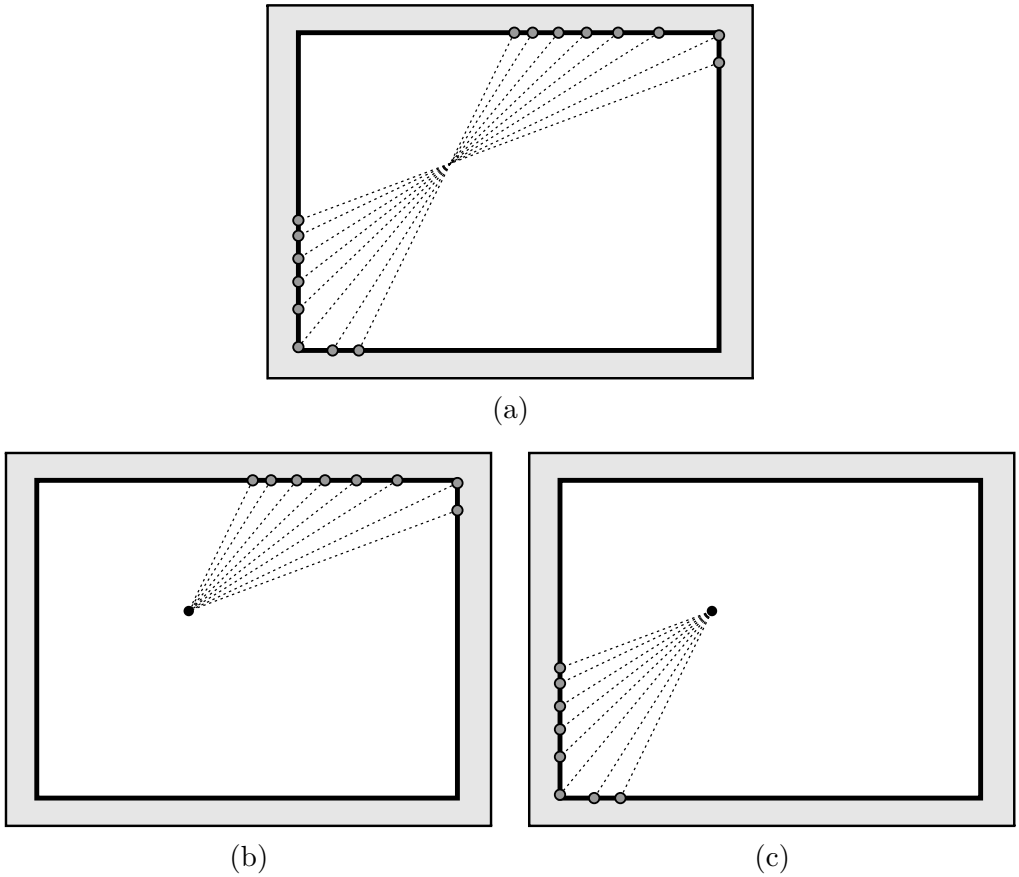


Figure 5.1: Separation of the propagation operator into a gathering and a projection operators. The propagation operator in (a) can be separated into a (b) gathering operator and a (c) projection operator.

Figure 5.2 shows how operators $\mathcal{Q}$, $\mathcal{E}$ and $\mathcal{R}$ are executed to reproduce the operator $\mathcal{G} : L \rightarrow L^*$. The separation of operator $\mathcal{G}$ into component operators $\mathcal{Q}$, $\mathcal{E}$ and $\mathcal{R}$ is useful because each component operator can then be individually mapped to a graphics hardware implementation. Also, it enables a reorganization of Veach's scattering-propagation formulation of the transport operator [50]. Using scattering and propagation operators,

the k-bounce light transport can be symbolically expressed as

$$\mathcal{T}^{(k)} = \mathcal{K}^{(k)} \circ \mathcal{G}^{(k)} \circ \mathcal{K}^{(k-1)} \circ \mathcal{G}^{(k-1)} \circ \cdots \circ \mathcal{K}^{(1)} \circ \mathcal{G}^{(1)} \circ \mathcal{K}^{(0)} \circ \mathcal{G}^{(0)}. \tag{5.5}$$

where $\mathcal{K}^{(k)}$ and $\mathcal{G}^{(k)}$ represent the k-th application of operators $\mathcal{K}$ and $\mathcal{G}$, respectively. Substituting equation 5.4 into 5.5 allows the k-bounce light transport to be expressed as

$$\begin{aligned}\mathcal{T}^{(k)} = \mathcal{K}^{(k)} \circ \mathcal{R}^{(k)} \circ \mathcal{E}^{(k)} \circ \mathcal{Q}^{(k)} \circ \mathcal{K}^{(k-1)} \circ \mathcal{R}^{(k-1)} \circ \mathcal{E}^{(k-1)} \circ \mathcal{Q}^{(k-1)} \circ \cdots \\ \circ \mathcal{K}^{(1)} \circ \mathcal{R}^{(1)} \circ \mathcal{E}^{(1)} \circ \mathcal{Q}^{(1)} \circ \mathcal{K}^{(0)} \circ \mathcal{R}^{(0)} \circ \mathcal{E}^{(0)} \circ \mathcal{Q}^{(0)}.\end{aligned} \tag{5.6}$$

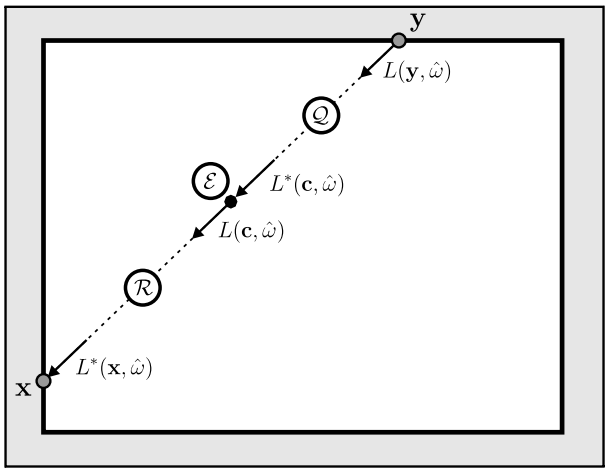


Figure 5.2: Operators $\mathcal{Q}$, $\mathcal{E}$ and $\mathcal{R}$ can be combined to reproduce the propagation operator $\mathcal{G}$ between two surface points $\mathbf{x}$ and $\mathbf{y}$.

Even though this formulation seems more complex, it allows us to regroup the scattering and propagation operators into a different organization. Combining operators in a

new grouping allows the definition of a new operator

$$\mathcal{P}^{(k)} = \mathcal{E}^{(k)} \circ \mathcal{Q}^{(k)} \circ \mathcal{K}^{(k-1)} \circ \mathcal{R}^{(k-1)}. \tag{5.7}$$

We will call operator $\mathcal{P} : L \rightarrow L$ the *pencil transport operator*. It describes the transport of radiance from a set of centers of projection to another center of projection. Operator $\mathcal{P}$ can also be interpreted as a sequence of simpler steps. The radiance transported after $k-1$ bounces and stored at a set of pencils is projected over the scene using operator $\mathcal{R}^{(k-1)}$. This radiance information is then scattered back in the direction of another pencil's center of projection through operator $\mathcal{K}^{(k-1)}$. The scattered radiance is then propagated to this pencil by the operator $\mathcal{Q}^{(k)}$, representing the incoming radiance transported after k bounces. Finally, the incident radiance is converted to exitant radiance via the operator $\mathcal{E}^{(k)}$. Figure 5.3 illustrates each of the operators composing the pencil transport operator.

Using the pencil transport operator $\mathcal{P}$, the k-order light transport in equation 5.5 can be rewritten as a sequence of pencil transports

$$\mathcal{T}^{(k)} = \mathcal{K}^{(k)} \circ \mathcal{R}^{(k)} \circ \mathcal{P}^{(k)} \circ \mathcal{P}^{(k-1)} \circ \cdots \circ \mathcal{P}^{(1)} \circ \mathcal{E}^{(0)} \circ \mathcal{Q}^{(0)}. \tag{5.8}$$

Equation 5.8 shows that the k-order light transport on a scene can be modeled as an initial gathering pass, followed by $k-1$ pencil transport passes, and a final projection-scattering pass to the camera. This formulation presents some immediate advantages.

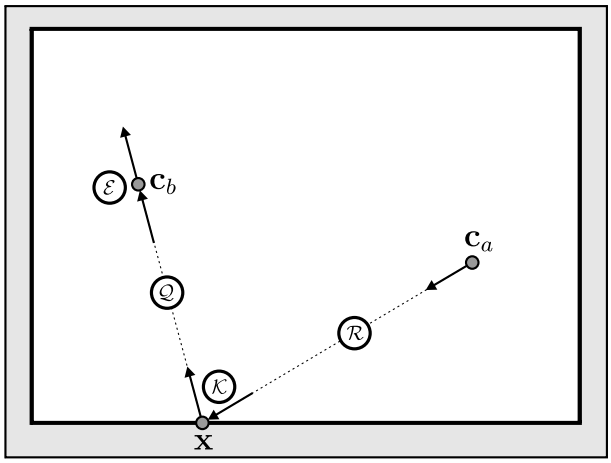


Figure 5.3: The pencil transport operator $\mathcal{P}$ is composed by projection $\mathcal{R}$, scattering $\mathcal{K}$, gathering $\mathcal{Q}$ and equivalence $\mathcal{E}$ operators.

One important advantage is that all radiance information, as well as other necessary information such as forward and backward depth information, only needs to be stored at pencils. Also, separating the final projection-scattering pass leads to a recursive definition of the k-order light transport operator:

$$\mathcal{T}^{(k+1)} = \mathcal{K}^{(k+1)} \circ \mathcal{R}^{(k+1)} \circ \mathcal{T}_P^{(k)} \tag{5.9}$$

$$\mathcal{T}_P^{(k)} = \mathcal{P}^{(k)} \circ \mathcal{T}_P^{(k-1)} \tag{5.10}$$

$$\mathcal{T}_P^{(0)} = \mathcal{E}^{(0)} \circ \mathcal{Q}^{(0)} \tag{5.11}$$

5.1 Light Transport Between Pencils

The pencil transport operator $\mathcal{P}$ can be used to create an efficient framework for light transport. In order to implement a pencil light transport algorithm, the operators that

define the pencil transport operator must be combined into a single equation. Fortunately, equation 5.7 can be combined and expressed as

$$\mathcal{P}^{(k)} = \mathcal{E}^{(k)} \circ \mathcal{Q}^{(k)} \circ \mathcal{K}^{(k-1)} \circ \mathcal{R}^{(k-1)}. \tag{5.12}$$

$$(\mathcal{P}g)(\mathbf{c}_o, \hat{\omega}_o) = \int_{\Omega_{\mathbf{x}}} f_r(\hat{\omega}_i, \mathbf{x}, \hat{\omega}_o)\ g(\mathbf{c}_i, \hat{\omega}_i)\ \cos\theta_i\ d\hat{\omega}_i. \tag{5.13}$$

for all surface points $\mathbf{x}$ satisfying the constraint

$$\mathbf{x} = h(\mathbf{c}_o, -\hat{\omega}_o) = h(\mathbf{c}_i, \hat{\omega}_i). \tag{5.14}$$

Constraint (5.14) is used to ensure that a surface point $\mathbf{x}$ is mutually visible from centers of projection $\mathbf{c}_i$ and $\mathbf{c}_o$, toward directions $\hat{\omega}_i$ (for rays leaving $\mathbf{c}_i$ and arriving at $\mathbf{x}$) and $-\hat{\omega}_o$ (for rays leaving $\mathbf{x}$ and arriving at $\mathbf{c}_o$), respectively. Since the ray casting function $h(\mathbf{x}, \hat{\omega})$ returns a unique solution, constraint 5.14 can be embedded into equation 5.12 using the visibility function:

$$(\mathcal{P}g)(\mathbf{c}_o, \hat{\omega}_o) = \int_{\Omega_{\mathbf{x}}} f_r(\hat{\omega}_i, \mathbf{x}, \hat{\omega}_o)\ g(\mathbf{c}_i, \hat{\omega}_i)\ V(\mathbf{x}, \mathbf{c}_i)\ \cos\theta_i\ d\hat{\omega}_i. \tag{5.15}$$

and the direction $\hat{\omega}_i$ can be computed explicitly as a function of $\mathbf{c}_o$, $\mathbf{c}_i$ and $\hat{\omega}_o$:

$$\hat{\omega}_i = \text{norm}\,(\mathbf{x} - \mathbf{c}_i) \tag{5.16}$$

where $\mathbf{x} = h(\mathbf{c}_o, -\hat{\omega}_o)$. Observe that the visibility function $V(\mathbf{x}, \mathbf{y})$ can be also defined using the ray casting function as

$$V(\mathbf{x}, \mathbf{y}) = \begin{cases} 1 & \text{if } \mathbf{x} = h(\mathbf{y}, \text{norm}\,(\mathbf{x} - \mathbf{y})) \\ \\ 0 & \text{otherwise} \end{cases} . \tag{5.17}$$

The pencil light transport operator in equation 5.15 can be applied to the exitant radiance field to perform the transport of radiance from a set of centers of projection towards the environment and then to a given center of projection. This transport between a set of pencils and a given pencil can be written as

$$L(\mathbf{c}_o, \hat{\omega}_o) = \int_{\Omega_\mathbf{x}} f_r(\hat{\omega}_i, \mathbf{x}, \hat{\omega}_o)\ L(\mathbf{c}_i, \hat{\omega}_i)\ V(\mathbf{x}, \mathbf{c}_i)\ \cos\theta_i\ d\hat{\omega}_i \tag{5.18}$$

This equation is only valid if centers of projection are distributed uniformly over the hemisphere Ω_x around the surface point $\mathbf{x}$. However, centers of projection are positioned at fixed points in free space and produce different hemispherical distributions for different surface points. This problem can be addressed by dividing the integrand with a density function that compensates for the non-uniform hemispherical distribution of centers of projection, resulting in

$$L(\mathbf{c}_o, \hat{\omega}_o) = \int_{\Omega_\mathbf{x}} \frac{f_r(\hat{\omega}_i, \mathbf{x}, \hat{\omega}_o)\ L(\mathbf{c}_i, \hat{\omega}_i)\ V(\mathbf{x}, \mathbf{c}_i)\ \cos\theta_i}{D(\mathbf{x}, \hat{\omega}_i)}\ d\hat{\omega}_i\,. \tag{5.19}$$

where $D(\mathbf{x}, \hat{\omega})$ is a distribution function of centers of projection around a point $\mathbf{x}$. We call function $D(\mathbf{x}, \hat{\omega})$ the *directional pencil density function* (DPDF).

Since pencils are distributed in free space, two or more centers of projection can be colinear to a given surface point $\mathbf{x}$. Therefore, centers of projections that are visible to the surface point $\mathbf{x}$ are transporting the same radiance contribution more than once to $\mathbf{x}$. That means the DPDF must also compensate for the non-uniform density of pencils along a given direction.

5.1.1 Directional Pencil Density function

The directional pencil density function $D(\mathbf{x}, \hat{\omega})$, describes the density of visible pencils around a given surface point $\mathbf{x}$ along a direction $\hat{\omega}$. In the limit when the number of pencils goes to infinity the hemispherical pencil distribution at a surface point $\mathbf{x}$ becomes uniform. On the other hand, the pencil density at a surface point $\mathbf{x}$ in a direction $\hat{\omega}$ is given by the distance between $\mathbf{x}$ and the closest surface point in direction $\hat{\omega}$:

$$D(\mathbf{x}, \hat{\omega}) = |\mathbf{x} - h(\mathbf{x}, \hat{\omega})| \,. \tag{5.20}$$

Equation 5.20 can be interpreted as an integration of Dirac pulses over the domain of points in free space:

$$D(\mathbf{x}, \hat{\omega}) = \int_{\mathcal{V}} \delta(|\hat{\omega} - \hat{\omega}_i|)\, V(\mathbf{x}, \mathbf{c}_i)\, d\mathbf{c}_i \tag{5.21}$$

For equation 5.19 to be valid, $D(\mathbf{x}, \hat{\omega})$ must integrate to unity over the hemisphere

around $\mathbf{x}$. This can be achieved dividing the pencil density in equation 5.21 by the volume of points visible to $\mathbf{x}$,

$$D(\mathbf{x}, \hat{\omega}) = \frac{\int\limits_{\mathcal{V}} \delta(|\hat{\omega} - \hat{\omega}_i|)\ V(\mathbf{x}, \mathbf{c}_i)\ d\mathbf{c}_i}{\int\limits_{\mathcal{V}} V(\mathbf{x}, \mathbf{c}_i)\ d\mathbf{c}_i} \tag{5.22}$$

In real applications, only a finite number of centers of projections are used, so a discrete version of equation 5.19 is used:

$$L(\mathbf{c}_o, \hat{\omega}_o) = \sum_{i=0}^{n_p} \frac{f_r(\hat{\omega}_i, \mathbf{x}, \hat{\omega}_o) L(\mathbf{c}_i, \hat{\omega}_i)\ V(\mathbf{x}, \mathbf{c}_i)\ \cos\theta_i}{D(\mathbf{x}, \hat{\omega}_i)} \tag{5.23}$$

where n_p is the number of pencils being used.

We must guarantee the partition of unity over visible sampled pencils while accordingly reflecting the correct distribution and density of pencils. The simplest approach to specify a DPDF over a set of visible center of projections is to use the discrete version of 5.22:

$$D(\mathbf{x}, \hat{\omega}) = \frac{\sum\limits_{i=0}^{n_p} \delta(|\hat{\omega} - \hat{\omega}_i|)\ V(\mathbf{x}, \mathbf{c}_i)}{\sum\limits_{i=0}^{n_p} V(\mathbf{x}, \mathbf{c}_i)}\,. \tag{5.24}$$

The reconstruction of a function $D(\mathbf{x}, \hat{\omega})$ representing the pencil density and hemispherical distribution at a given surface point $\mathbf{x}$ can be treated as a spherical sparse data interpolation problem. Several methods have been studied in order to perform interpolation of sparse data sampled over a sphere, including spherical harmonics [129, 130], spherical triangulations [131], spherical splines [132], and spherical radial basis func-

tions [133, 134].

Spherical harmonics [129, 130] have become a common technique in computer graphics to store hemispherical representations over a sphere [135–138]. Spherical harmonics can be seen as an analogy of polynomial interpolation performed over a spherical topology. However, since spherical harmonics are direct analogs of bivariate polynomials, they tend to oscillate due to their global nature and are highly susceptible to ringing at function discontinuities. In our application, oscillations that produce zero or negative values must be avoided. Also, they are more suitable for representing smooth functions [139].

It is also possible to convert an interpolation problem over a sphere to an interpolation problem defined on a rectangle [132, 140]. However, methods that use this approach usually present difficult problems at the poles [132, 141]. Common solutions for this issue are based on the use of periodic trigonometric B-splines [132] or spherical triangulations [131].

These techniques are not the most suitable for direct implementation of the directional pencil density function using graphics hardware. Spherical triangulations as well as spherical spline interpolations require knowledge about local connectivity over the sphere. This connectivity must be determined dynamically for each surface point since it changes for different surface points for the same set of pencils. Spherical harmonics is a good option for final representation of a smooth function over the sphere, such as the results of an integration with a smooth BRDF, but they are not appropriate for representation of discontinuous functions, such as incoming radiance fields. Also, spherical harmonics coefficients are too expensive to be computed on the fly during a fragment program. In the

context of pencil light transport, a more appropriate technique for interpolating sparse data on a sphere consists of using radial basis functions.

Radial basis functions (RBF) are usually referred to as the composition of a univariate positive function with a distance function [142, 143]. For a given sample i positioned at $\mathbf{x}_i$, the corresponding radial basis function $\Phi_i(\mathbf{x})$ is given by

$$\Phi_i(\mathbf{x}) = \varphi(|\mathbf{x} - \mathbf{x}_i|) \tag{5.25}$$

where $|\mathbf{x} - \mathbf{x}_i|$ is the distance between $\mathbf{x}$ and $\mathbf{x}_i$, and $\varphi(r)$ is called the *basic function*. Examples of basic functions commonly used are

$$\varphi(r) = r^2 \log r \tag{5.26}$$

$$\varphi(r) = e^{-cr^2} \tag{5.27}$$

$$\varphi(r) = \sqrt{r^2 + c^2} \tag{5.28}$$

Basic function 5.26 is the *thin-plate spline* and is commonly used for fitting smooth functions. Basic function 5.27 is the Gaussian bump and is mostly used in neural network applications. Basic function 5.28 is the multiquadric function and is commonly used for fitting topographical data.

An approximation of a function using a set of samples can then be achieved by linear combination of radial basis functions. A simple and straightforward method for sparse

data interpolation using radial basis functions is the Shepard method [144]. The Shepard method basically defines a set of basis functions that form a partition of the unity by explicitly normalizing an arbitrary set of basis functions $\Phi_i(\mathbf{x})$:

$$\widehat{\Phi}_i(\mathbf{x}) = \frac{\Phi_i(\mathbf{x})}{\sum_{j=0}^{n} \Phi_j(\mathbf{x})} \tag{5.29}$$

and then approximating the function by the sum

$$\widetilde{f}(\mathbf{x}) = \sum_{i=0}^{n} f(\mathbf{x}_i)\widehat{\Phi}_i(\mathbf{x}_i)\,. \tag{5.30}$$

Interpolation methods based on radial basis functions are among the most effective for dealing with sparse data sets, specially two-dimensional functions [145]. To approximate the directional pencil density function we propose the use of spherical radial basis functions, with the Shepard approximation method.

A *spherical radial basis function* (SRBF) is an analogy of a radial basis function on a sphere [133]. However, spherical radial basis functions are specified with respect to the geodesic distance[1] instead of the Euclidean distance. The general form of a spherical radial basis function is given by

$$\Phi_i(\hat{\omega}) = \varphi(d(\hat{\omega}, \hat{\omega}_i)) \tag{5.31}$$

[1]Geodesic distance is defined as the length of the (shortest) great circle arc connecting two points on a sphere

where $d(\hat{\omega}, \hat{\omega}_i)$ is the geodesic distance between the points $\hat{\omega}$ and $\hat{\omega}_i$.

Light [146] has shown that basic functions constructed for standard radial basis functions can be mapped to the spherical form, so that appropriate conversions of known RBF basic functions can be used in the spherical form. Recently, spherical Gaussians [147] and cosine functions [148] have been proposed as basic functions for SRBF interpolation.

Spherical radial basis functions are particularly suitable for the implementation of a directional pencil density function since each directional sample is treated individually. We propose the use of spherical radial basis functions to approximate the directional pencil density function, using Phong-like cosine lobes as a basic functions:

$$D_i(\mathbf{x}) = \sum_{j=0}^{n_p} V(\mathbf{x}, \mathbf{c}_j)\ \cos_+^{\beta_j} \theta_{ij} \tag{5.32}$$

where θ_{ij} is angle between the directions from surface point $\mathbf{x}$ to the centers of projection $\mathbf{c}_i$ and $\mathbf{c}_j$, and β_j is an exponent that defines the width of the cosine lobes towards each center of projection $\mathbf{c}_j$.

Our choice of a Phong-like cosine lobe as the basic function for SRBF interpolation is based on the fact that they have good support for direct implementation in graphics hardware. Also, making the exponents β_j proportional to the mean distance between centers of projection provides the correct DPDF in the limit as the number of pencils goes to infinity. Figure 5.4 shows how cosine lobes can be combined in order to create a continuous directional pencil density function at surface point $\mathbf{x}$.

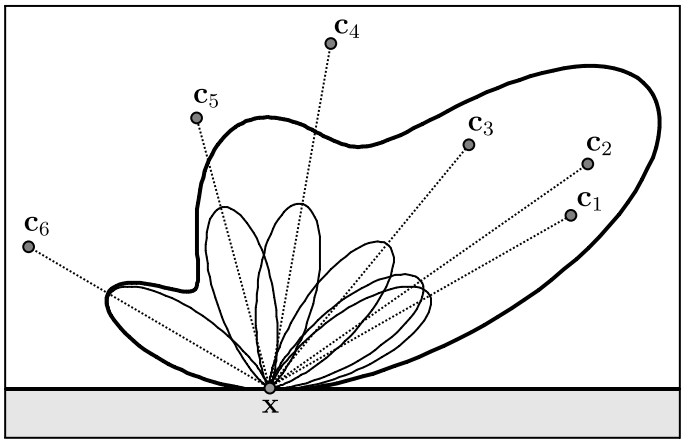


Figure 5.4: Directional pencil density function at a point $\mathbf{x}$.

Finally, in order to enforce the partition of unity constraint, we perform an explicit normalization of the directional pencil density function, as with the Shepard method [144]:

$$D(\mathbf{x}, \hat{\omega}_i) = \frac{V(\mathbf{x}, \mathbf{c}_i) D_i(\mathbf{x})}{\sum_{j=0}^{n_p} V(\mathbf{x}, \mathbf{c}_j) D_j(\mathbf{x})} \tag{5.33}$$

An additional benefit of using this formulation comes from the fact that each function $D_i(\mathbf{x})$ can be represented as a set of directional data at each pencil's center of projection. Since the DPDF at a surface point $\mathbf{x}$ only accounts for centers of projection that are visible to $\mathbf{x}$, the function $D_i(\mathbf{x})$ relative to a center of projection $\mathbf{c}_i$ can be precomputed and stored at each pencil i and parameterized using a direction $\hat{\omega}_i$.

In fact, this approach to compute the DPDF is very similar to rendering an image with the Phong lighting model, using multiple light sources, followed by a normalization step. Compared to the alternatives, this technique of sparse data interpolation is simple,

fast, and has a good mapping onto current graphics hardware.

5.2 Summary

This chapter has described the pencil light transport method for simulating global illumination. This method is based on the transfer of radiance over the environment using a set of pencil objects. We construct the pencil transport operator that performs the transport of radiance from a set of pencils to a given pencil. We also show how the pencil light transport operator can be specified as a non-uniform sampling problem at surface points and suggest a technique to compensate for the non-uniformity of centers of projection. Some extensions to the basic pencil light transport algorithm are discussed in the conclusions, where we will also discuss some of the tradeoffs involved in our choice of sparse data interpolant.

Chapter 6

Implementation and Results

This chapter describes some implementation details for the pencil light transport algorithm, including the process for the creation of pencils and how we iterate between sets of pencils. We analyze the results from our pencil light transport implementation, including octahedral projection. We describe some possible refinements to the standard algorithm such as iteration using variable sets of pencils, and alternative methods for selecting centers of projections.

6.1 Basic Implementation

Chapter 5 presented the main idea of the pencil light transport method as well as an important operator for light transport which we call the pencil transport operator. A pencil transport operator is used to transfer radiance from a set of pencils toward a

given pencil's center of projection. A standard implementation of a pencil light transport algorithm can be separated into five stages:

1. Select a number of centers of projection;
2. Construct a pencil for each center of projection;
3. Perform a initial gathering operation for each pencil;
4. Iterate transfer of radiance information between pencils;
5. Perform a final projection-scatter operation towards the camera.

Each of these steps has a number of variations with different tradeoffs, which we will discuss in the following.

6.1.1 Selecting Centers of Projection

This step consists of selecting a number of centers of projection distributed over the environment. These centers of projection define the set of pencil objects that are used to transport radiance over the environment.

A simple approach for distributing centers of projection over the environment is to draw random positions inside environment boundaries and reject points that are invalid, such as points inside objects. However, more elaborate sampling strategies can improve convergence and reduce artifacts. We discuss some alternative strategies for selecting centers of projection in section 6.2.2.

6.1.2 Creating Pencils

Once a number of centers of projections have been selected, pencil objects are constructed by creating textures for radiance, depth information and directional density function $D_i(\hat{\omega})$ at each pencil's center of projection. The combination of these three textures together with the center of projection constitutes a pencil object. Since we are using omni-directional pencils, each texture is parameterized over a sphere, similar to environment maps. Depth information textures are only dependent on the scene geometry relative to each pencil's center of projection and so these maps only need to be computed once for each pencil. Textures containing non-normalized directional pencil density functions depend only on the scene geometry and on the other centers of projection, and do not change when radiance information is iterated between pencils. These textures therefore only need to be computed once for each pencil. At this step radiance textures are allocated, but not initialized. The initialization of radiance textures is performed during the initial gathering operation step.

Depth textures can be obtained by rendering the environment to an environment map centered at each pencil's center of projection and storing at the corresponding pixel the distance between each fragment coordinate in world space and the center of projection. The non-normalized directional pencil density function texture for a given pencil p can be obtained using the same procedure, but running algorithm 6.1 for each rasterized fragment. The coordinates of each fragment in world space is given by $\mathbf{x}$ and all other parameters have the semantics described in chapter 5.

Algorithm 6.1 Algorithm for computing the pencil density function at center of projection $\mathbf{c}_p$.

compute $\hat{\omega}_p \leftarrow \text{norm}\,(\mathbf{c}_p - \mathbf{x})$
for $i = 0$ to n_p **do**
  compute $\hat{\omega}_i \leftarrow \text{norm}\,(\mathbf{c}_i - \mathbf{x})$
  compute $\cos_+ \theta_i = \max(\hat{\omega}_p \cdot \hat{\omega}_i, 0)$
  accumulate $V(\mathbf{c}_i, \mathbf{x})\ \cos_+^{\beta_i} \theta_i$ in the basis function texture
end for

The overall computational cost associated with the acquisition of depth textures and pencil density function textures for a set of n pencils is of order $O(n)$ for depth textures and of order $O(n^2)$ for pencil density function textures.

6.1.3 Initial Gathering Operation

The main purpose of the initial gathering step is to initialize the radiance textures stored at each pencil. In its simplest form it is equivalent to rendering of the emitted radiance L_e from the environment's light sources to the radiance environment map for each pencil. However, a more efficient approach to the initial gathering operation is to render the direct illumination from the scene towards every pencil. Using the direct illumination instead of the emitted radiance in the initial gathering step allows us to start the iteration process with a considerably larger number of radiance samples. Direct illumination approximations can be efficiently computed on current graphics hardware, so this optimization can significantly increase the performance of a pencil light transport implementation. The only difference that must be accounted for when using the direct illumination for the initial gathering operation is that the iteration stage starts at the light bounce compo-

nent $\mathcal{T}L_e$ instead L_e. Starting the iteration using the direct illumination component corresponds to the next event estimation optimization described in section 2.4 for path tracing.

In order to estimate the direct illumination at surface points from the scene in the presence of area sources and large numbers of light sources, we generate a set of light samples distributed over light sources and determine the amount of power to be emitted by each light sample. The direct illumination at a given surface point is then estimated using area integration over the light samples. We choose this technique due to its simplicity and robustness. However, other techniques can be used in order to estimate the direct illumination [63]. Based on a set of light samples, the initial gathering operation for each pencil p can be described using algorithm 6.2. Algorithm 6.2 is executed for each fragment with world space coordinates $\mathbf{x}$ and local surface normal $\hat{\mathbf{n}}_{\mathbf{x}}$. Each light sample's attributes are treated as global parameters and a light sample i is positioned at $\mathbf{l}_i$, and has orientation $\hat{\mathbf{n}}_i$ and emitted radiance E_i.

Algorithm 6.2 Algorithm to compute the initial gathering operation for pencil p.

compute $\hat{\omega}_p \leftarrow \text{norm}\,(\mathbf{c}_p - \mathbf{x})$
for all light sample i **do**
  compute $\hat{\omega}_i \leftarrow \text{norm}\,(\mathbf{l}_i - \mathbf{x})$
  compute $\cos_+ \theta_i = \max(\hat{\omega}_i \cdot \hat{\mathbf{n}}_{\mathbf{x}}, 0)$
  compute $\cos_+ \theta_l = \max(-\hat{\omega}_i \cdot \hat{\mathbf{n}}_i, 0)$
  compute $r \leftarrow |\mathbf{l}_i - \mathbf{x}|$
  accumulate $f_r(\hat{\omega}_p, \mathbf{x}, \hat{\omega}_i)\; V(\mathbf{l}_i, \mathbf{x})\; E_i\; \dfrac{\cos_+ \theta_i \cos_+ \theta_l}{r^2}$ in the radiance texture
end for

6.1.4 Iterating Radiance Between Pencils

This step is responsible for transferring the radiance between two sets of pencils through the scene geometry. This can be done by applying the pencil transport operator over all pencils in a set of pencils. One pencil transport operator application accounts for the propagation of radiance from one set of pencils towards the environment and then to a given target pencil. This step assumes that radiance textures are already initialized, as well as depth and pencil density function textures. The implementation of the pencil light transport operator from a set of pencils to a given pencil p can be done by performing a modified version of a scene rendering using multiple point light sources with shadowing. The modification includes using centers of projection as point light sources and using the radiance textures to obtain the corresponding radiance arriving at surface points. The radiance information arriving at a surface point $\mathbf{x}$ from a given pencil can be obtained by performing a texture lookup into the associated radiance texture. Algorithm 6.3 describes this procedure in more detail.

Each iteration between the pencils of a set represents one light bounce in the environment. To account for one light bounce over the environment for all n pencils of a set, algorithm 6.3 is executed n times, one for each pencil in the set, as illustrated in figure 6.1. In order to make a pencil's radiance textures converge to the steady-state radiance distribution at its center of projection, we take into account the fact that the

Algorithm 6.3 Algorithm to compute the radiance transport from a set of pencils to a pencil p.

compute $\hat{\omega}_p \leftarrow \text{norm}\,(\mathbf{c}_p - \mathbf{x})$
for all pencils i **do**
  compute $\vec{\omega}_i \leftarrow \mathbf{c}_i - \mathbf{x}$
  use $\vec{\omega}_i$ to lookup radiance, depth and pencil density function from pencil i
  compute $\hat{\omega}_i \leftarrow \text{norm}\,(\vec{\omega}_i)$
  evaluate the BRDF $f_r(\hat{\omega}_p, \mathbf{x}, \hat{\omega}_i)$ at fragment position $\mathbf{x}$
  evaluate visibility term $V(\mathbf{x}, \mathbf{c}_i)$
  compute $\cos\theta_i \leftarrow \max(\hat{\mathbf{n}}_x \cdot \hat{\omega}_i, 0)$
  accumulate $\dfrac{f_r(\hat{\omega}_p, \mathbf{x}, \hat{\omega}_p) L(\mathbf{c}_i, \hat{\omega}_i) \cos\theta_i}{D_i(\mathbf{x})}$ in the RGB channels
  accumulate $D_i(\mathbf{x})$ in the alpha channel
end for
multiply RGB components by the alpha channel

accumulated radiance transported after k light-surface bounces can be expressed as

$$\sum_{i=1}^{k+1} \mathcal{T}^i L_e = \mathcal{T}\left(\sum_{i=1}^{k} \mathcal{T}^i L_e\right) + \mathcal{T} L_e\,. \tag{6.1}$$

Using the decomposition in equation 6.1, it is possible to obtain the accumulated radiance after $k+1$ bounces by performing a light bounce iteration over all pencils in a set (that are assumed to have radiance textures with the accumulated radiance after k light bounces) and adding the direct illumination (which we can cache from the first step).

The main advantage of this approach is that only radiance textures need to be modified during the light transport simulation. Since centers of projection are fixed, depth information is unchanged. Also, since the set of pencils used for the simulation is also unchanged, there is no need to recompute directional pencil density functions.

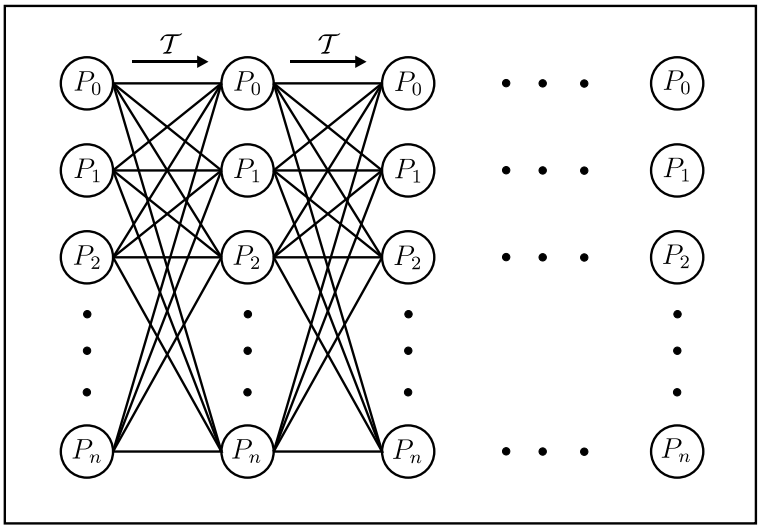


Figure 6.1: Iteration over a set of pencils.

One problem with this approach is that it requires either a direct illumination rendering pass for each pencil for every light bounce rendering or an extra radiance texture for each pencil to store the direct illumination component. Also, iterating over a fixed set of pencils performs correlated radiance transports and may produce structured visible artifacts in the final image. These artifacts are especially noticeable on undersampled regions (regions containing surfaces that are visible to only a small number of centers of projection.)

This pencil iteration procedure has computational cost of order $O(kn^2)$ to compute the global illumination solution for k light bounces using a set of n pencils. Since radiance is transferred between each pair of pencils of the set, the cost to compute each light bounce contribution is $O(n^2)$. The cost to compute pencil density function textures is $O(n^2)$ and to compute the depth textures is $O(n)$, but they only need to be computed once per

Texture	**Complexity**
Radiance	$O(kn^2)$
Pencil Density	$O(n^2)$
Depth	$O(n)$

Table 6.1: Order of complexity using a fixed set of n pencils.

light transport simulation. Nonetheless, the overall complexity of the iteration process is $O(kn^2)$. Table 6.1 summarizes the orders of complexity for the computation of individual sets of textures.

6.1.5 Final Projection-Scatter Operation

The final projection-scatter operation consists of projecting the radiance information stored at each pencil towards the environment and them scattering it to the camera position. This operation can be performed by executing algorithm 6.3, but using the camera position instead of the center of projection $\mathbf{c}_p$, and rendering using a perspective projection instead of an environment map projection. In fact, the camera can be seen as a specialization of a pencil object, which has a coverage defined by a rectangle and only implements the gathering operation.

6.2 Refinements

The performance and image quality of a pencil light transport implementation can be improved by considering some refinements over the basic algorithm. It is possible to improve overall performance using a different approach for iterating radiance between

pencils. Also, more elaborate techniques for selecting centers of projection can lead to improvements image quality and convergence rates.

6.2.1 Iterating Radiance between Pencils

The iteration process described in section 6.1.4 performs the simulation of a light bounce transport by projecting the radiance from a set of pencils towards the environment and gathering the scattered radiance back into the same set of pencils. However, a known property of the light transport equation is that the total amount of radiance propagated over the environment decreases geometrically after each light bounce iteration. Also, it is a common behaviour for the residual radiance field distribution to become smoother as the number of light bounces increases and therefore it will have lower variance. Based on these properties, an alternative iteration approach is to use sets of pencils with different sizes to perform the light transport at each light bounce. For each new light bounce iteration, only a fraction of number of centers of projection created for the previous light bounce are required, defining a sequence of set of pencils with geometrically decreasing sizes.

Using a geometrically decreasing number of pencils for each light bounce iteration has several advantages over the original algorithm. Since the number of centers of projections decreases geometrically after each light bounce iteration, the computational cost to compute the radiance textures for all pencil sets after a sufficiently large number of light bounces is $O(n^2)$ instead of $O(kn^2)$, where n is the size of the initial pencil set. However,

Texture	**Complexity**
Radiance	$O(n^2)$
Pencil Density	$O(n^2)$
Depth	$O(kn)$

Table 6.2: Order of complexity using geometrically decreasing number of pencils.

this approach requires the initialization of depth and pencil density function textures for all new pencils. That means the cost to compute the depth textures becomes $O(kn)$ and the cost to compute the pencil density function textures becomes to $O(n^2)$. Even so, the overall order of complexity of the pencil light transport algorithm using this approach is $O(n^2)$, since $k < n$. Table 6.2 summarizes the order of complexity for the computation of each individual set of textures.

One problem with creating new pencils for each light bounce iteration is that memory requirements to store all pencils objects increases significantly as the number of light bounces increases. In order to reduce the memory requirements to store pencil objects, we can accumulate partial light bounce results and discard the previous set of pencils processed on each light bounce iteration.

Let $\mathcal{V}_i$ be a set of pencils used to simulate the light transport bounce of order $i+1$, after a final projection-scatter operation, as illustrated in figure 6.2. For each light bounce, a projection-scatter operation is applied to the set of pencils $\mathcal{V}_i$ and the resulting camera image is accumulated. Then a light bounce iteration is performed to propagate radiance from a set of pencils $\mathcal{V}_i$ to a set of pencils $\mathcal{V}_{i+1}$. After these three steps are completed, the set of pencils $\mathcal{V}_i$ is not necessary for the rest of the simulation and can be safely

discarded. Algorithm 6.4 describes the pencil iteration algorithm using sets of pencils with geometrically decreasing sizes. After this algorithm is performed, the accumulation texture contains the global illumination solution after $k+2$ light-surface bounces.

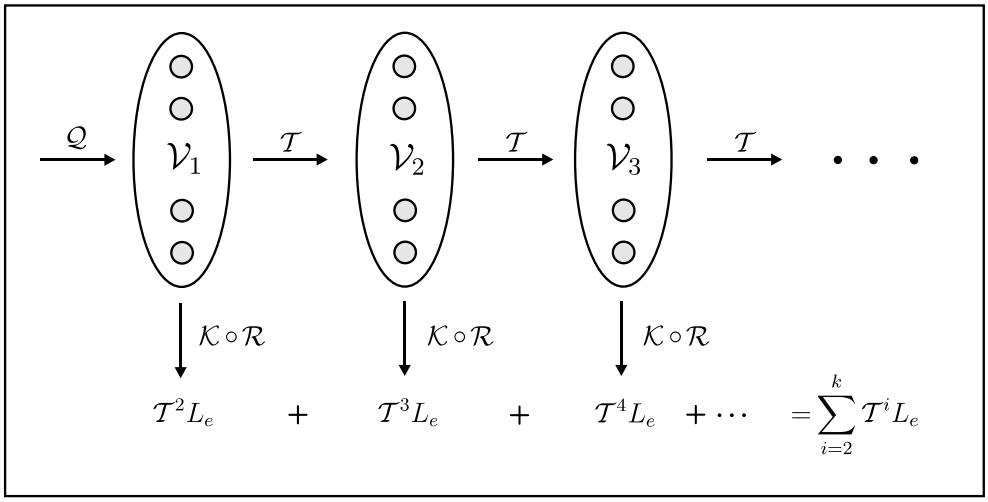


Figure 6.2: Pencil transport using partial results accumulation. After each set of pencils $\mathcal{V}_i$ is acquired, their radiance information is projected to the camera and accumulated. After that the pencil set $\mathcal{V}_i$ can be deleted.

Using algorithm 6.4 significantly reduces the memory requirements to implement a pencil light transport algorithm that uses different sets of pencils for each light bounce.

6.2.2 Selecting Centers of Projection

The use of an explicitly normalized directional pencil density function allows for pencils to have a non-uniform distribution over the environment. This property enables us to exploit various center of projection placement strategies to improve convergence and to reduce visual artifacts in the global illumination solution. We have not tested all possible

Algorithm 6.4 Algorithm to simulate light transport by accumulating partial results.

```
Create a accumulation screen texture
Initialize the accumulation texture with the sum of direct illumination and emissivity
Select and create initial set of pencils V_0
Perform initial gathering pass for V_0
for i = 0 to k do
    Perform a projection-scatter step over V_i and accumulate the result
    Select and create new set of pencils V_{i+1}
    Perform a light bounce iteration from V_i to V_{i+1}
    Release V_i
end for
Perform a projection-scatter step over V_k and accumulate the result
```

placement strategies, but in this section we discuss some desirable properties for centers of projection, as well as strategies to position centers of projection based on these properties.

The general goal of an efficient global illumination algorithm is to transport as much radiance to as many surface points as possible for a given computational budget. In the context of a pencil light transport algorithm, this goal is equivalent to selecting centers of projection on regions that maximize the visibility coverage of the scene by a set of pencils, as well as regions that maximize the overall power transported through a set of pencils. However, the global illumination solution must be known in order to determine regions that maximize visibility and power throughput properties.

The power throughput at a given pencil can be computed using graphics hardware by mipmapping the pencil's radiance texture. A fragment program can be used then to render the top level mipmap textures from all pencils to a relatively small texture that can be read back into main CPU memory and used to select the new centers of projection. The visibility component can be computed using a similar procedure, but by rendering

the scene using only the visibility term.

A solution for the problem of estimating regions with high visibility and power throughput is to compute an *illumination profile* of the scene, and use information from this profile to drive the selection of centers of projection. The use of profiling techniques to determine suitable regions for selecting samples is a common approach in multigrid techniques [149]. The illumination profile can be estimated progressively by performing a dense sampling of centers of projection during the propagation of the direct illumination component, and refining the illumination profile as new centers of projection are sampled after each new light bounce. Using an illumination profile to determine properties of the radiance field dynamically after each light bounce and selecting centers of projection on regions where visibility and power throughput are high would enable us to reduce the initial number of pencils by a larger factor.

However, determining a full illumination profile could be computationally expensive. An alternative solution to the use of an illumination profile is to use *probing rays* to select centers of projection. For a given light bounce computation, a small number of rays could be traced from light sources, and centers of projection positioned on each sampled ray. We name these rays *light probing rays*. For instance, centers of projection could be positioned at the midpoint between the ray's origin and intersection point. For each additional light bounce computation, a fraction of the initial rays are continued and centers of projection positioned on the new sampled rays. The probing rays selected to be continued can be determined based on the accumulated importance, similar to the

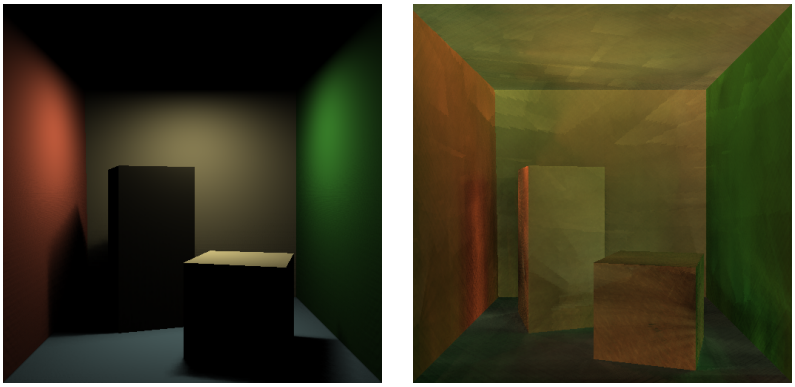
importance used in the Russian roulette technique [64]. Additionally, probing rays can be traced from the camera position towards the scene. We call these rays *camera probing rays*. Camera probing rays can be used to select a different set of centers of projection performing the same procedure used to select centers of projection on light probing rays.

The estimates computed using the sets of pencils selected from light probing rays and camera probing rays can be combined in order to provide a more accurate estimate for the light bounce component. The estimates can be combined using a multiple importance estimator [59]. In fact, it is possible to combine estimates accounting for different illumination effects using a multiple importance estimator. While we have not implemented these strategies, we plan to make their investigation the focus of future work.

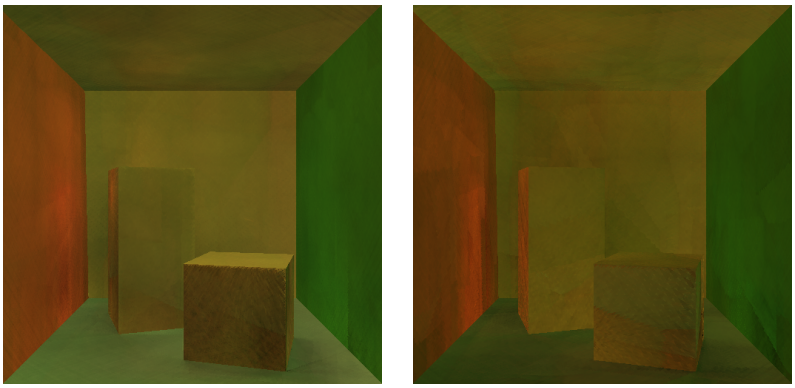
6.3 Results

This section presents some results from our current implementation of the pencil light transport algorithm, which uses a random center of projection placement strategy. We analyze the convergence behaviour as well as the main artifacts resulting from a pencil light transport algorithm. We also present some results and sampling rate analysis for the octahedral parameterization.

Figure 6.3 shows the light bounce components and the resulting global illumination solution of a Cornell Box rendered using pencil light transport using 100, 50 and 25 pencils for the computation of the second, third and forth light bounce contributions, respectively. Each light bounce component is scaled in order to provide a more visible



(a) (b)



(c) (d)

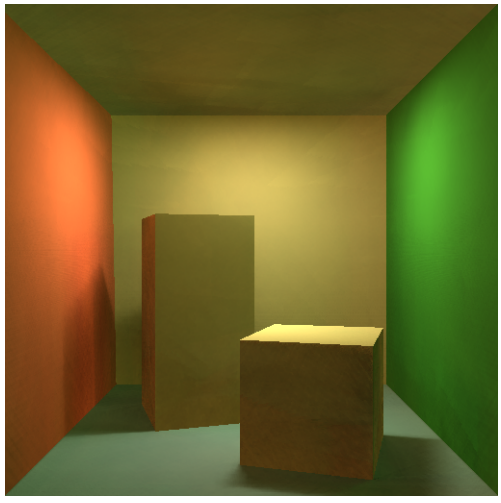
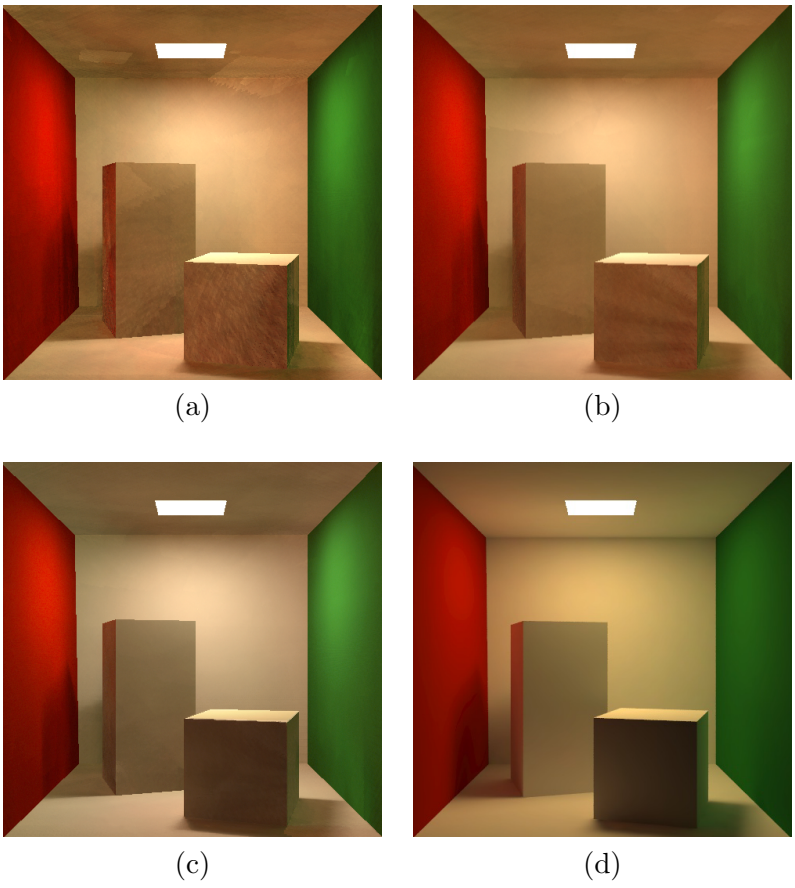


Figure 6.3: Illumination components in a Cornell box: (a) Direct illumination (50 light samples), (b) second light bounce (100 pencils), (c) third light bounce (50 pencils), (d) forth light bounce (25 pencils) and (e) final solution.

image of the overall light distribution of each light bounce component.

Figure 6.4 shows images generated using increasing number of centers of projection and a reference image for the Cornell box. Images are rendered using sets of 128, 64 and 32 pencils for the computation of the second light bounce component and with 0.7 pencil set decrease rate. The reference image is the official Cornell box reference, and it was generated by a spectral global illumination rendering system using material properties represented over the full colour spectra.



(a) (b)

(c) (d)

Figure 6.4: Images are rendered using different sizes for the initial set: (a) 32 pencils, (b) 64 pencils, and (c) 128 pencils. (d) Reference image.

Figure 6.5 shows difference images between the corresponding images 6.4(a), (b) and (c) and the reference image 6.4(d). Figure 6.5 illustrates the convergence behaviour of the pencil light transport algorithm as the number of pencils increases as well as regions that present high error. Difference images are computed using distances in CIE La*b* color space. More elaborate metrics are available [150–152], but the La*b* distance provides enough accuracy for a visual analysis of artifacts and convergence behaviour of the algorithm [153]. From figures 6.4 and 6.5 it is possible to observe artifacts localized systematically on some regions. Figure 6.6 identifies some these artifacts for further analysis.

The artifact exemplified in region 1 of Figure 6.6 happens mostly along corners and can be visually observed as inaccurate color bleeding. This artifact results from the reduced number of centers of projections available close to corners, where it happens that most of the radiance transport responsible for the color bleeding effect is being transported. A possible solution for this issue is the use of transillumination planes [122] along concave edges. However, the use of transillumination planes requires analysis of the environment geometry and a more elaborate sampling mechanism. In fact, transillumination planes can be considered a special case of a pencil with the center of projection at infinity

Region 2 contains artifacts caused by low visibility of pencils. This artifact appears on shadows or dark areas of the environment, where few pencils can be seen from surface points. This issue causes regions to have darker illumination results as well as inaccurate color bleeding effects. This artifact can be addressed performing a denser sampling of

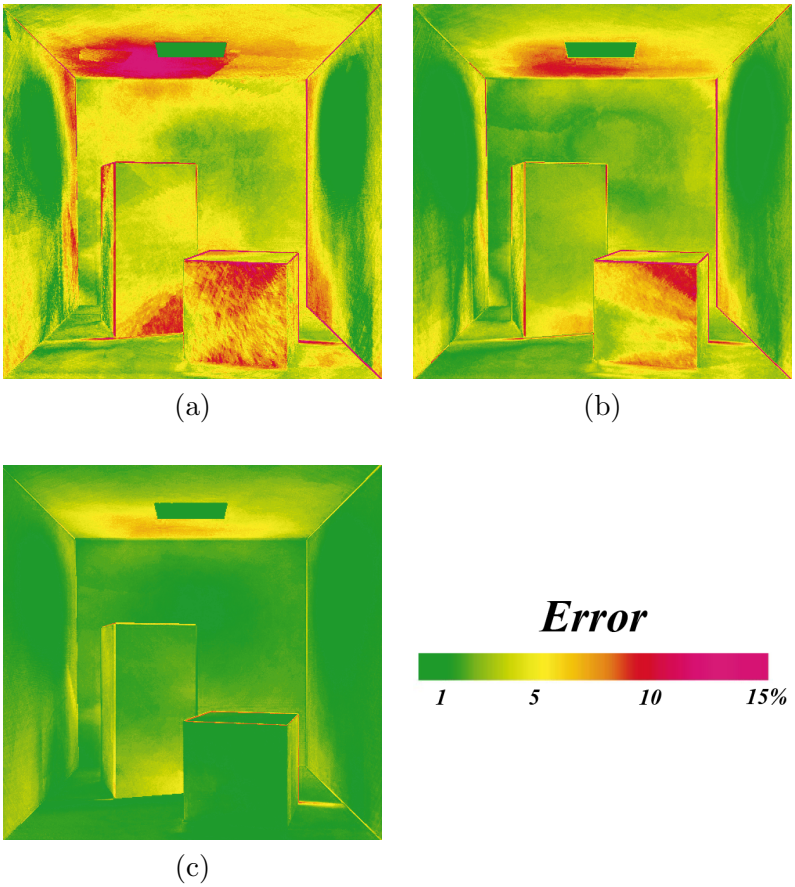


Figure 6.5: Difference images, in La*b* color space, between images 6.4(a), (b) and (c) and the reference image 6.4(d).

centers of projection close to these regions in order to transfer more light from regions presenting higher sampling densities. Region 3 presents a low convergence rate due to the combination of the artifacts described for regions 1 and 2, namely inaccurate color bleeding and low pencil visibility, and can be addressed using a combination of the procedures describe before.

Region 4a and 4b show aliasing artifacts due to insufficient resolution of the depth map used for projecting radiance information. These errors are perceived as light leakage

on penumbra boundaries, as shown in region 4b. When centers of projection are relatively close to objects, similar aliasing artifacts can happen, as in region 4b. Since these issues are mainly related to the shadow mapping technique, they can be solved using a different visibility approach, such as shadow volumes [154, 155], or using alternative shadow mapping techniques [84, 156]. Anisotropic filtering can also be used to reduce aliasing artifacts [157], specially the one shown in region 4b. It should be noted that any GPU-based global illumination algorithm that depends on shadow maps for visibility will have to deal with the limited resolution available in this shadowing technique.

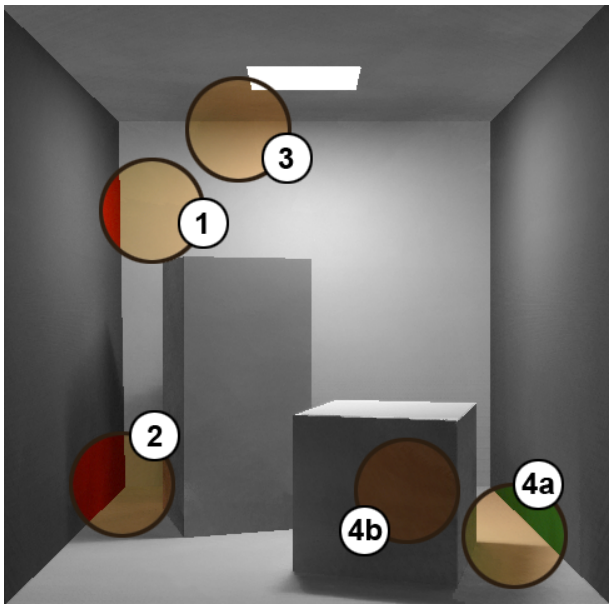


Figure 6.6: Common artifacts present in images rendered by a pencil light transport algorithm.

Table 6.3 shows the timings of individual stages of our pencil light transport implementation using pencil sets with different sizes. Table 6.4 shows the timings for the

transport of radiance from a set of pencils (source pencils) to another set of pencils (target pencils). Finally, table 6.5 shows the timings for the acquisition of depth maps and for the rendering of the direct illumination component to the camera. Processing times were obtained using a Pentium IV 2.0 Ghz, 512Mb RAM, AGP 8x using a NVidia 6800GT graphics card with 256Mb of video memory. The test environment is the Cornell box, rendered with 512×512 resolution.

	12 Pencils	**25 Pencils**	**50 Pencils**	**100 Pencils**
Sampling and creation	0.048	0.100	0.472	0.657
Initial gather	0.247	0.462	1.427	2.297
Depth texture render	0.086	0.182	0.676	1.479
DPDF texture render	0.306	1.326	5.167	17.85
Final projection-scatter	0.007	0.014	0.025	0.045

Table 6.3: Timings, in seconds, for some stages of our pencil light transport implementation.

Source pencils	**Target pencils**	**Time (s)**
25	12	0.31
25	25	0.76
50	25	2.98
100	50	10.50

Table 6.4: Timings, in seconds, for transporting radiance from a set of source pencils to a set of target pencils.

Rendering step	**Number of light samples**	**Time (s)**
Light depth map acquisition	50	0.32
Direct illumination rendering	50	0.015

Table 6.5: Timings for direct illumination step.
Timings, in seconds, for direct illumination step.

Figure 6.7 describes the observed error convergence of our current pencil light transport implementation. Error for a given rendered image is estimated using root mean

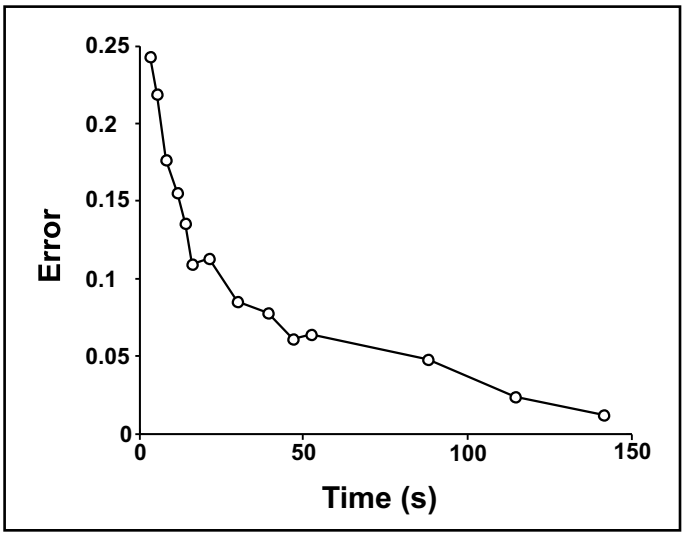


Figure 6.7: Observed error convergence of the pencil light transport implementation. Error measures are evaluated using root mean square of La*b* distances to the reference image in figure 6.4(d).

square of the La*b* distances to the reference image in figure 6.4(d). We observe that the algorithm presents a convergence bias. The convergence bias is caused mainly due to aliasing resulting from the limited number of sampled centers of projection. This issue is expected to happen since the final set of pencils can be interpreted as a sampled representation of the radiance field. Additional bias is introduced by the truncation of the rendering equation, and can be reduced using probabilistic termination. Other minor sources of bias are due to specific features included in the reference image but not for accounted in our rendered images. In particular, the reference image used slightly different material properties represented over a higher-dimensional colour space, which we could not duplicate in our implementation.

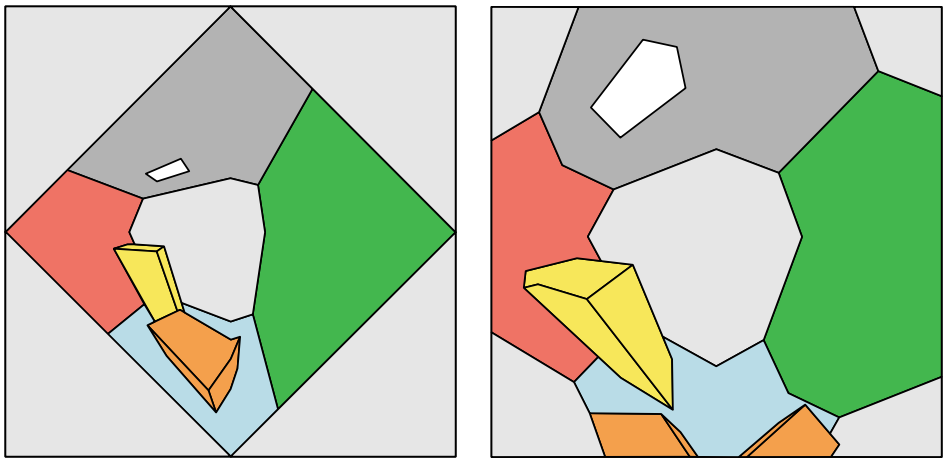
6.3.1 Octahedral Projection

The pencil light transport algorithm does not require the use of a specific environment mapping parameterization. However, the proposed octahedral environment mapping projection presents some advantages for implementing a pencil light transport algorithm over other common environment mapping parameterizations. In general, octahedral parameterization is advantageous when a large number of complete environment map lookups are required during a single rendering pass.

Using octahedral parameterization it is possible to perform 4 environment map lookups using 18 instructions. The same computation using a parabolic parameterization requires 35 instructions, and 29 instructions using a cube parameterization stored using tiled two-dimensional textures. Storing cubemaps as tiled two-dimensional textures are necessary in order to be able to use more than 16 cubemaps on a single pass on current graphics hardware. It is possible to use dependent texture lookups in order to improve the computation of a cubemap lookup to 23 instructions, but octahedral environment mapping still presents faster texture lookup code. Using the same computing configuration as before, we are able to render the Cornell box scene performing 128 octahedral environment map lookups in 11.37 miliseconds.

Figure 6.8 shows some examples of the Cornell box rendered using an octahedral parameterization. In figure 6.8(a) the center of projection is positioned on the front boundary of the box, so the backward hemisphere contains no geometry. Figure 6.8(b) illustrates the parameterization distortion as the center of projection is moved forward

inside the scene geometry.



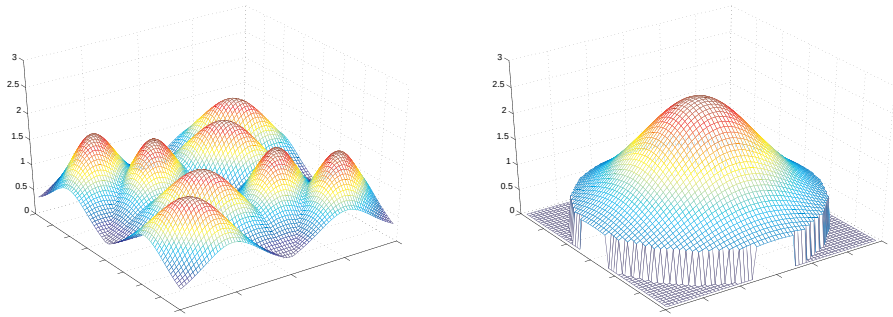
(a) (b)

Figure 6.8: Examples of the deformations produced by the octahedral parameterization. (a) Center of projection positioned on the front boundary of the environment and (b) Center of projection inside the environment.

Octahedral parameterization also presents a suitable sampling distribution for a pencil light transport implementation. Even though the parabolic parameterization has a smaller ratio between the largest and the smallest sampling rates, the octahedral projection has a more uniform overall distribution around the ideal sampling rate. Since pencils' directional data are entirely projected towards the environment, it is desirable for a parameterization to be able to store directional data as uniformly as possible. Figure 6.9 shows the sampling distributions for the parabolic and octahedral parameterizations. Table 6.6 shows the sampling ratios and sampling variations of parabolic, cube and octahedral parameterizations.

Parameterization	Sampling ratio	Sampling variation
Parabolic	4.00	0.44
Cube	5.19	0.27
Octahedral	5.19	0.26

Table 6.6: Sampling ratios and sampling rate variations of the parabolic, cube and octahedral parameterizations.



(a) (b)

Figure 6.9: Sampling rate distributions of different parameterizations: (a) Octahedral projection and (b) Parabolic projection.

6.4 Discussion

We have demonstrated that the pencil light transport algorithm can be used to simulate global illumination effects such as indirect illumination, soft shadows and color bleeding. These effects are important in applications such as architectural and lighting design. The majority of the light transport simulation process is done using optimized graphics hardware features, and only the selection of centers of projection is done by the CPU. The pencil light transport algorithm does not require the environment to be stored in a specific data structure or to be preprocessed in any way. In fact, the algorithm does not make any assumptions about the method used to render the scene geometry.

Most light transport algorithms are based on the idea of reducing the number of illumination samples (rays, path, patches, etc.) and use more elaborate integration methods to find global illumination solutions. In contrast, the pencil light transport method is based on the use of a simple integration method while transporting a large number of illumination samples. Simpler integration methods can be implemented using current instruction-limited fragment programs and large number of illumination samples can be propagated using a hardware-based environment map implementation. Since graphics processors have been presenting an increase in performance considerably larger than main system processors over last few years, pencil light transport is a potentially useful approach to render scenes with dynamic and unrestricted geometry and lighting conditions at interactive rates.

However, the pencil light transport algorithm is not without its problems. Pencil light transport is susceptible to undersampling issues. Artifacts related to these issues normally appear in regions containing dark shadows or in corners. Also, the algorithm has some bias issues that arise mostly when an insufficient number of pencils is used to represent the radiance field. Another major source of bias is the deterministic truncation of the rendering equation. This bias also occurs in light and path tracing algorithms that do not implement a bias compensation technique, such as Russian roulette. It should be noted that other popular global illumination solution techniques, such as photon mapping and meshed radiosity, are also biased. In future work, it would be useful to try and improve the visual quality of images generated using pencil light transport, for instance by blurring

the radiance fields gathered and propagated by the pencils. Such blurring is not physically valid, but might improve perceived visual quality, just as interpolation of radiance samples in meshed radiosity suppresses the appearance of discretization artifacts.

6.5 Summary

This chapter described the main aspects of our implementation of the pencil light transport algorithm. We described the main stages composing a pencil light transport algorithm, as well as some refinements to improve the convergence rate and image quality. We also presented some results showing the behaviour of the pencil light transport algorithm with respect to a varying number of pencils and analyzed the most important artifacts of our pencil light transport implementation.

Chapter 7

Conclusions

We have presented a novel method for handling global illumination problems, called *pencil light transport*. The new method performs the transport of radiance over an environment using a set of centers of projection distributed non-uniformly over the environment. Centers of projection are used to create pencil objects that are used as the base mechanism for performing the light transport between surfaces. A *pencil object* consists of a center of projection together with some associated directional data. Pencil objects can be used to perform two main operations: *gathering* and *projection*. These two operations are combined over multiple pencils to propagate radiance information over a scene. Both pencil operations can be implemented using optimized graphics hardware features such as two dimensional textures, triangle rasterization, and shader programs.

We presented a new operator that accounts for the light transport between two sets of pencils, and showed how the light transport equation can be formulated as a sequence of

pencil transports, preceeded by an initial gathering step and followed by a final projection-scatter step. We described how this formulation can be implemented on current graphics hardware, as well as some refinements for the pencil transport algorithm in order to improve convergence rates and image quality. A new environment map parameterization has been proposed that allows for the transfer of radiance from a large number of centers of projections to a given surface point.

There are several extensions that could be implemented as extensions to our system in order to handle more general illumination conditions and more complex environments, and also to improve visual appearance. Even though the directional pencil density functions used in our implementation provide good results, we would like to test different methods for estimating the pencil density in the scene in order to provide better convergence and image quality. Also, our current implementation specifies the pencils' cosine exponents in an ad-hoc manner. It would be useful to derive a more automatic method for computing suitable values for these parameters.

A natural extension to pencil light transport is to perform the iteration between pencils using a hierarchical structure, similar to methods previously used for radiosity [75, 77]. Since centers of projection tend to be localized on regions with difficult illumination conditions, it is possible to define groups of pencils based on the proximity of their centers of projections. The hierarchy can be constructed so that groups of pencils with highest proximity are in the lowest levels of the hierarchy. Given a hierarchy of groups of pencil, each group of pencils is iterated locally, starting with groups at lower levels and then

progressively iterating groups on higher levels.

The idea of transporting information using pencil objects can also be used in different applications to simulate specific illumination effects. For instance, pencil objects might be used to simulate subsurface scattering on highly translucent materials or to render glossy refractive objects. These approximations could be accomplished by positioning centers of projection inside objects and perturbing the direction vectors used during gathering and projection operations based on the object's surface normals.

Another potential application for using pencil objects is to interactively visualize high quality global illumination solutions. Pencil objects essentially represent two dimensional samples of a five dimensional radiance field. When performing the final projection-scattering step, no assumption is made about the method used to acquire the two dimensional radiance samples stored at each pencil. This allows us to select a number of center of projections over an environment and using another global illumination approach to acquire the radiance field at each center of projection. During the visualization of the environment only the final projection-scattering step is performed. Additionally, depending on the size of environment being visualized, only a subset of the total number of pencils may be necessary for particular regions and groups of pencils can be added or removed when moving from one region to another.

In conclusion, we have presented a novel approach to global illumination that runs directly on graphics hardware. There are, however, many ways in which our core idea could be extended.

Bibliography

[1] C. M. Goral, K. E. Torrance, D. P. Greenberg, and B. Battaile, "Modelling the interaction of light between diffuse surfaces," in *Proceedings of SIGGRAPH 1984*, pp. 213–222, July 1984.

[2] J. T. Kajiya, "The rendering equation," in *Proceedings of SIGGRAPH 1986*, pp. 143–150, August 1986.

[3] S. Pattanaik and S. Mudur, "Efficient potential equation solutions for global illumination computation," *Computers & Graphics*, vol. 17, no. 4, pp. 387–396, 1993.

[4] E. P. Lafortune and Y. D. Willems, "Bi-directional Path Tracing," in *Proceedings of Third International Conference on Computational Graphics and Visualization Techniques*, (Alvor, Portugal), pp. 145–153, 1993.

[5] E. Veach and L. Guibas, "Bidirectional estimators for light transport," in *Eurographics Workshop on Rendering 1994*, pp. 147–162, June 1994.

[6] H. W. Jensen, "Global illumination using photon maps," in *Proceedings of the 7th Eurographics Workshop on Rendering*, (Porto, Portugal), pp. 21–30, June 1996.

[7] I. Wald, C. Benthin, M. Wagner, and P. Slusallek, "Interactive rendering with coherent ray tracing," *Computer Graphics Forum (Proceedings of EUROGRAPHICS 2001*, vol. 20, no. 3, pp. 153–164, 2001.

[8] T. D. Alan Chalmers and E. Reinhard, eds., *Practical Parallel Rendering.* AK Peters, Ltd., 2002.

[9] B. Walter, G. Drettakis, and S. Parker, "Interactive rendering using the render cache," in *Proceedings of the 10th Eurographics Workshop on Rendering*, (Granada, Spain), June 1999.

[10] K. Bala, B. Walter, and D. P. Greenberg, "Combining edges and points for interactive high-quality rendering," *ACM Transactions on Graphics, Proceedings of ACM SIGGRAPH 2003*, vol. 22, pp. 631–640, July 2003.

[11] G. J. Ward and P. Heckbert, "Irradiance gradients," in *Proceedings of the Eurographics Workshop on Rendering 1992*, pp. 85–98, May 1992.

[12] M. Simmons and C. H. Séquin, "Tapestry: A dynamic mesh-based display representation for interactive rendering," in *Eurographics Workshop on Rendering 2000*, pp. 329–340, June 2000.

[13] P. Tole, F. Pellacini, B. Walter, and D. P. Greenberg, "Interactive global illumi-

nation in dynamic scenes," *ACM Transactions on Graphics*, vol. 21, pp. 537–546, July 2002.

[14] G. W. Larson and M. Simmons, "The holodeck ray cache: An interactive rendering system for global illumination in non-diffuse environments," *ACM Transactions on Graphics*, vol. 18, pp. 361–368, October 1999.

[15] G. Greger, P. Shirley, P. M. Hubbard, and D. P. Greenberg, "The irradiance volume," *IEEE Computer Graphics & Applications*, vol. 18, pp. 32–43, March 1998.

[16] K. Bala, J. Dorsey, and S. Teller, "Radiance interpolants for accelerated bounded-error ray tracing," *ACM Transactions on Graphics*, vol. 18, pp. 213–256, August 1999.

[17] K. Dmitriev, S. Brabec, K. Myszkowski, and H.-P. Seidel, "Interactive global illumination using selective photon tracing," in *Proceedings of the 13th Eurographics Workshop on Rendering*, pp. 25–36, June 2002.

[18] G. Drettakis and F. X. Sillion, "Interactive update of global illumination using a line-space hierarchy," in *Proceedings of SIGGRAPH 1997*, pp. 57–64, August 1997.

[19] S. Venkatasubramanian, "The graphics card as a stream computer," in *Workshop on Management and Processing of Data Streams*, (San Diego, California), June 2003.

[20] A. Barsi and G. Jakab, "Stream processing in global illumination," in *Proceedings of 8th Central European Seminar on Computer Graphics*, April 2004.

[21] K. Moreland and E. Angel, "The FFT on a GPU," in *Proceedings of the Workshop on Graphics Hardware 2003*, pp. 112–119, July 2003.

[22] E. S. Larsen and D. K. McAllister, "Fast matrix multiplies using graphics hardware," in *Proceedings of Supercomputing 2001*, (Denver), November 2001.

[23] Á. Moravánszky, *ShaderX2 : Shader Programming Tips and Tricks with DirectX 9.0*, ch. Dense Matrix Algebra on the GPU, p. 400. Wordware Publishing, Inc., 2003.

[24] J. Bolz, I. Farmer, E. Grinspun, and P. Schröder, "Sparse matrix solvers on the GPU: Conjugate gradients and multigrid," *ACM Transactions on Graphics, Proceedings of ACM SIGGRAPH 2003*, vol. 22, pp. 917–924, July 2003.

[25] M. Finch, *GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics*, ch. Effective Water Simulation from Physical Models, p. 816. Addison-Wesley Professional, 2004.

[26] Y. L.-X. Liu and E. Wu, "Real-time 3D fluid simulation on GPU with complex obstacles," in *Proceedings of Pacific Graphics 2004*, October 2004.

[27] R. Strzodka, *Hardware Efficient PDE Solvers in Quantized Image Processing*. PhD thesis, University of Duisburg-Essen, 206.

[28] A. Keller, "Instant radiosity," in *Proceedings of SIGGRAPH 1997*, pp. 49–56, August 1997.

[29] L. Szirmay-Kalos and W. Purgathofer, "Global ray-bundle tracing with hardware acceleration," in *Proceedings of the 9th Eurographics Workshop on Rendering*, (Porto, Portugal), pp. 247–258, June 1998.

[30] L. Szirmay-Kalos and W. Purgathofer, "Global ray-bundle tracing with infinite number of rays," *Computers and Graphics*, 1999.

[31] T. J. Purcell, I. Buck, W. R. Mark, and P. Hanrahan, "Ray tracing on programmable graphics hardware," *ACM Transactions on Graphics*, vol. 21, pp. 703–712, July 2002.

[32] F. Lavignotte and M. Paulin, "Scalable photon splatting for global illumination," in *Proceedings of the International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia*, (Melbourne, Australia), pp. 1–11, February 2003.

[33] B. D. Larsen and N. J. Christensen, "Simulating photon mapping for real-time applications," in *Eurographics Symposium on Rendering*, (Norrköping, Sweden), June 2004.

[34] T. J. Purcell, C. Donner, M. Cammarano, H. W. Jensen, and P. Hanrahan, "Photon

mapping on programmable graphics hardware," in *Graphics Hardware 2003*, (San Diego, California), pp. 41–50, July 2003.

[35] R. Siegel, J. R. Howell, and J. Howell, *Thermal Radiation Heat Transfer*. Taylor & Francis, Inc., 4 ed., 2004.

[36] A. S. Glassner, *Principles of Digital Image Synthesis*. Morgan Kaufmann, 1 ed., 1995.

[37] P. Dutré, P. Bekaert, and K. Bala, *Advanced Global Illumination*. Natick, MA: AK Peters, Ltd., 1 ed., July 2003.

[38] J. M. Hammersley and D. C. Handscomb, *Monte Carlo Methods*. London: Methuen & Co., 1964.

[39] M. H. Kalos and P. A. Whitlock, *Monte Carlo Methods*. New York: John Wiley & Sons, 1986.

[40] D. Shreiner, M. Woo, J. Neider, and T. Davis, *OpenGL Programming Guide: Version 1.4*. Addison-Wesley Professional, 4 ed., 2003.

[41] M. Segal and K. Akeley, *The OpenGL Graphics System: A Specification (Version 2.0)*. Silicon Graphics, Inc., 2004.

[42] J. Kessenich, D. Baldwin, and R. Rost, *The OpenGL Shading Language*. 3Dlabs, Inc. Ltd., 2004.

[43] R. Fernando, ed., *GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics*. Addison-Wesley Professional, 2004.

[44] M. Pharr and R. Fernando, eds., *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*. Addison-Wesley Professional, 2005.

[45] M. McCool and S. Du Toit, *Metaprogramming GPUs with Sh*. AK Peters, Ltd., 2004.

[46] C. Decusatis, ed., *Handbook of Applied Photometry*. AIP Press, 1997.

[47] Bureau International des Poids et Mesures, "Système international d'unités," Sèvres, France, 1998.

[48] B. N. Taylor, ed., *Guide for the Use of the International System of Units (SI)*. National Institute of Standards and Technology, 1995.

[49] F. E. Nicodemus, J. C. Richmond, and J. J. Hsia, *Geometrical Considerations and Nomenclature for Reflectance*. National Bureau of Standards, 1977.

[50] E. Veach, *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, Stanford University, December 1997.

[51] R. R. Lewis, "Making shaders more physically plausible," in *Fourth Eurographics Workshop on Rendering*, (Paris, France), pp. 47–62, 1993.

[52] P. S. Shirley, *Physically Based Lighting Calculations for Computer Graphics*. PhD thesis, University of Illinois at Urbana-Champaign, 1990.

[53] N. Dunford and J. T. Schwartz, *Linear Operators, General Theory*. Wiley-Interscience, 1988.

[54] J. Arvo, *Analytic Methods for Simulated Light Transport*. PhD thesis, Yale University, 1995.

[55] A. Fournier, "From local to global illumination and back," in *Eurographics Rendering Workshop 1995*, pp. 127–136, June 1995.

[56] D. B. Kirk and J. Arvo, "Unbiased sampling techniques for image synthesis," in *Proceedings of SIGGRAPH 1991*, vol. 25, pp. 153–156, July 1991.

[57] D. P. Mitchell, "Consequences of stratified sampling in graphics," in *Proceedings of SIGGRAPH 1996*, pp. 277–280, August 1996.

[58] M. Steigleder and M. McCool, "Generalized stratified sampling using the Hilbert curve," *Journal of Graphics Tools*, vol. 8, no. 3, pp. 41–47, 2003.

[59] E. Veach and L. J. Guibas, "Optimally combining sampling techniques for Monte Carlo rendering," in *Proceedings of SIGGRAPH 95*, pp. 419–428, August 1995.

[60] T. Whitted, "An improved illumination model for shaded display," *Communications of the ACM*, vol. 23, pp. 343–349, June 1980.

[61] R. L. Cook, T. Porter, and L. Carpenter, "Distributed ray tracing," in *Proceedings of SIGGRAPH 1984*, vol. 18, pp. 137–145, July 1984.

[62] N. Chin and S. Feiner, "Fast object-precision shadow generation for area light sources using BSP trees," in *Proceedings of the symposium on Interactive 3D graphics 1992*, (New York, NY, USA), pp. 21–30, ACM Press, 1992.

[63] P. Shirley, C. Wang, and K. Zimmerman, "Monte Carlo techniques for direct lighting calculations," *ACM Transactions on Graphics*, vol. 15, pp. 1–36, January 1996.

[64] J. Arvo and D. B. Kirk, "Particle transport and image synthesis," in *Proceedings of SIGGRAPH 1990*, vol. 24, pp. 63–66, August 1990.

[65] S. N. Pattanaik and S. P. Mudur, "Computation of global illumination by Monte Carlo simulation of the particle model of light," in *Thrid Eurographics Workshop on Rendering*, pp. 71–83, May 1992.

[66] L. Szirmay-Kalos, "Stochastic methods in global illumination: State of the art report," Tech. Rep. TR-186-2-98-23, Technical University of Budapest, 1994.

[67] J. Talbot, D. Cline, and P. Egbert, "Importance resampling for global illumination," in *Eurographics Symposium on Rendering 2005*, 2005.

[68] E. Veach and L. J. Guibas, "Metropolis light transport," in *Proceedings of SIGGRAPH 97*, pp. 65–76, August 1997.

[69] J. Arvo, "Backward ray tracing," in *SIGGRAPH 1986 Developments in Ray Tracing seminar notes*, vol. 12, August 1986.

[70] P. Shirley, "A ray tracing method for illumination calculation in diffuse-specular scenes," in *Proceedings of Graphics Interface 1990*, 1990.

[71] H. W. Jensen, "Photon maps in bidirectional Monte Carlo ray tracing of complex objects," *Computer & Graphics*, vol. 19, no. 2, pp. 215–224, 1995.

[72] J. Bentley, "Multidimensional binary search trees used for associative searching," *Communication of the ACM*, vol. 18, pp. 509–517, September 1975.

[73] H. W. Jensen and P. H. Christensen, "Efficient simulation of light transport in scenes with participating media using photon maps," in *Proceedings of SIGGRAPH 1998*, pp. 311–320, July 1998.

[74] T. Nishita and E. Nakamae, "Continuous tone representation of three-dimensional objects taking account of shadows and interreflection," in *Proceedings of SIGGRAPH 1985*, vol. 19, pp. 23–30, July 1985.

[75] P. Hanrahan, D. Salzman, and L. Aupperle, "A rapid hierarchical radiosity algorithm," in *Proceedings of SIGGRAPH 1991*, pp. 197–206, July 1991.

[76] B. E. Smits, J. Arvo, and D. H. Salesin, "An importance-driven radiosity algorithm," in *Proceedings of SIGGRAPH '92*, pp. 273–282, July 1992.

[77] L. Aupperle and P. Hanrahan, "A hierarchical illumination algorithm for surfaces with glossy reflection," in *Proceedings of SIGGRAPH 1993*, pp. 155–162, August 1993.

[78] P. Shirley, *Graphics Gems II*, ch. Radiosity via Ray Tracing, p. 672. Morgan Kaufmann, 1991.

[79] L. Neumann, M. Feda, M. Kopp, and W. Purgathofer, "A new stochastic radiosity method for highly complex scenes," in *Eurographics Workshop on Rendering 1994*, pp. 201–213, 1994.

[80] L. Neumann, W. Purgathofer, R. F. Tobler, A. Neumann, P. Eliás, M. Feda, and X. Pueyo, "The stochastic ray method for radiosity," in *Eurographics Workshop on Rendering 1995*, pp. 206–218, June 1995.

[81] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan, "Brook for GPUs: stream computing on graphics hardware," *ACM Transactions on Graphics, Proceedings of ACM SIGGRAPH 2004*, vol. 23, pp. 777–786, August 2004.

[82] M. Segal, C. Korobkin, R. van Widenfelt, J. Foran, and P. E. Haeberli, "Fast shadows and lighting effects using texture mapping," in *Proceedings of SIGGRAPH 1992*, pp. 249–252, July 1992.

[83] L. Williams, "Casting curved shadows on curved surfaces," in *Proceedings of SIGGRAPH '78*, pp. 270–274, August 1978.

[84] M. Stamminger and G. Drettakis, "Perspective shadow maps," *ACM Transactions on Graphics*, vol. 21, pp. 557–562, July 2002.

[85] S. Brabec, T. Annen, and H.-P. Seidel, "Shadow mapping for hemispherical and omnidirectional light sources," in *Proceedings of Computer Graphics International 2002*, July 2002.

[86] P. Haeberli and M. Segal, "Texture mapping as a fundamental drawing primitive," in *Fourth Eurographics Workshop on Rendering*, pp. 259–266, June 1993.

[87] N. Greene, "Environment mapping and other applications of world projections," *IEEE Computer Graphics and Applications*, vol. 6, pp. 21–29, November 1986.

[88] W. Heidrich and H.-P. Seidel, "View-independent environment maps," in *1998 SIGGRAPH Eurographics Workshop on Graphics Hardware*, pp. 39–46, August 1998.

[89] W. Heidrich and H.-P. Seidel, "Realistic, hardware-accelerated shading and lighting," in *Proceedings of SIGGRAPH 1999*, pp. 171–178, August 1999.

[90] T. Akenine-Moller and E. Haines, eds., *Real-Time Rendering*. AK Peters, Ltd., 2002.

[91] M. F. Cohen and D. P. Greenberg, "The hemi-cube: A radiosity solution for complex environments," in *Proceedings of SIGGRAPH 1985*, vol. 19, pp. 31–40, Aug. 1985.

[92] M. Stamminger, A. Scheel, X. Granier, F. Perez-Cazorla, G. Drettakis, and F. X. Sillion, "Efficient glossy global illumination with interactive viewing," in *Graphics Interface '99*, pp. 50–57, June 1999.

[93] M. Stamminger, A. Scheel, X. Granier, F. Perez-Cazorla, G. Drettakis, and F. X. Sillion, "Efficient glossy global illumination with interactive viewing," *Computer Graphics Forum*, vol. 19, pp. 13–25, March 2000.

[94] X. Granier, G. Drettakis, and B. Walter, "Fast global illumination including specular effects," in *Proceedings of 11th the Eurographics Workshop on Rendering*, pp. 47–58, June 2000.

[95] X. Granier and G. Drettakis, "Incremental updates for rapid glossy global illumination," *Computer Graphics Forum*, vol. 20, no. 3, pp. 268–277, 2001.

[96] B. Walter, G. Drettakis, and D. P. Greenberg, "Enhancing and optimizing the render cache," in *Proceedings of the 13th Eurographics Workshop on Rendering*, pp. 37–42, June 2002.

[97] J. Zaninetti, X. Serpaggi, and B. Péroche, "A vector approach for global illumination in ray tracing," *Computer Graphics Forum*, vol. 17, no. 3, pp. 149–158, 1998.

[98] R. Crespin and B. Péroche, "Lights vectors for a moving observer," in *Proceedings of the 12th International Conference in Central Europe on Computer Graphics,*

Visualization and Computer Vision, (Plzen, Czech Republic), pp. 89–96, February 2004.

[99] G. Fournier and B. Péroche, "Multi-mesh caching and hardware sampling for progressive and interactive rendering," in *Proceedings of the 13th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, (Plzen, Czech Republic), pp. 63–70, February 2005.

[100] M. Nijasure, "Interactive global illumination on the GPU," Master's thesis, University of Central Florida, November 2003.

[101] K. Bala, J. Dorsey, and S. Teller, "Conservative radiance interpolants for ray tracing," in *Proceedings of the 7th Eurographics Workshop on Rendering*, (Porto, Portugal), pp. 257–268, June 1996.

[102] K. Bala, J. Dorsey, and S. Teller, "Interactive ray-traced scene editing using ray segment trees," in *Proceedings of the 10th Eurographics Workshop on Rendering*, (Granada, Spain), pp. 31–44, June 1999.

[103] H. Niederreiter, *Random Number Generation and Quasi-Monte Carlo Methods*. Philadelphia: CBMS-NSF Regional Conference Series in Applied Mathematics, 1992.

[104] H. E. Rushmeier, D. R. Baum, and D. E. Hall, "Accelerating the hemi-cube al-

gorithm for calculating radiation form factors," *ASME Journal of Heat Transfer*, vol. 113, pp. 1044–1047, November 1991.

[105] G. Coombe, M. J. Harris, and A. Lastra, "Radiosity on graphics hardware," in *Proceedings of Graphics Interface 2004*, (London, Canada), pp. 161–168, May 2004.

[106] N. A. Carr, J. D. Hall, and J. C. Hart, "GPU algorithms for radiosity and subsurface scattering," in *Graphics Hardware 2003*, (San Diego, California), pp. 51–59, July 2003.

[107] N. A. Carr and J. C. Hart, "Meshed atlases for real-time procedural solid texturing," *ACM Transactions on Graphics*, vol. 21, pp. 106–131, April 2002.

[108] T. J. Purcell, *Ray Tracing on a Stream Processor*. PhD thesis, Stanford university, March 2004.

[109] N. A. Carr, J. D. Hall, and J. C. Hart, "The ray engine," in *Graphics Hardware 2002*, (Saarbrücken, Germany), pp. 37–46, September 2002.

[110] M. Christen, "Ray tracing on GPU," Master's thesis, University of Applied Sciences Basel, January 2005.

[111] F. Karlsson, "Ray tracing fully implemented on programmable graphics hardware," Master's thesis, Chalmers University of Technology, 2004.

[112] D. Cohen and Z. Sheffer, "Proximity clouds, an acceleration technique for 3D grid traversal," *The Visual Computer*, vol. 11, no. 1, pp. 27–38, 1994.

[113] T. Möller and B. Trumbore, "Fast, minimum storage ray-triangle intersection," *Journal of Graphics Tools*, vol. 2, no. 1, pp. 21–28, 1997.

[114] F. Lavignotte, *Calcul de l'Éclairement Global par Estimation de Densité et par une Approche Image*. PhD thesis, Université Paul Sabatier, July 2003.

[115] W. Stürzlinger and R. Bastos, "Interactive rendering of globally illuminated glossy scenes," in *Proceedings of the 8th Eurographics Workshop on Rendering*, pp. 93–102, June 1997.

[116] F. Lavignotte and M. Paulin, "A new approach of density estimation for global illumination," in *Proceedings of the 10th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, (Plzen, Czech Republic), pp. 263–269, February 2002.

[117] J.-M. Hasenfratz, M. Lapierre, N. Holzschuch, and F. Sillion, "A survey of real-time soft shadows algorithms," *Computer Graphics Forum*, vol. 22, pp. 753–774, December 2003.

[118] B. D. Larsen and N. J. Christensen, "Optimizing photon mapping using multiple photon maps for irradiance estimates," in *Proceedings of the 11th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, (Plzen, Czech Republic), February 2003.

[119] M. Stamminger, J. Haber, H. Schirmacher, and H.-P. Seidel, "Walkthroughs with

corrective texturing," in *Procedings of the 11th Eurographics Workshop on Rendering*, pp. 377–390, 2000.

[120] R. Bastos, K. E. H. III, W. Wynn, and A. Lastra, "Increased photorealism for interactive architectural walkthroughs," in *Proceedings of the ACM Symposium on Interactive 3D Graphics 1999*, pp. 183–190, April 1999.

[121] R. Bastos, *Superposition Rendering: Increased Realism for Interactive Walkthroughs*. PhD thesis, University of North Carolina at Chapel Hill, 1999.

[122] S.-K. László, F. Tibor, N. László, and C. Balázs, "An analysis of quasi-Monte Carlo integration applied to the transillumination radiosity method," *Computer Graphics Forum*, vol. 16, no. 3, 1998.

[123] W. Heidrich, K. Daubert, J. Kautz, and H.-P. Seidel, "Illuminating micro geometry based on precomputed visibility," in *Proceedings of ACM SIGGRAPH 2000*, pp. 455–464, July 2000.

[124] K. Daubert, W. Heidrich, J. Kautz, J.-M. Dischler, and H.-P. Seidel, "Efficient light transport using precomputed visibility," *IEEE Computer Graphics and Applications*, pp. 28–37, May 2003.

[125] M. Shinya, T. Takahashi, and S. Naito, "Principles and applications of pencil tracing," in *Computer Graphics (SIGGRAPH '87 Proceedings)*, vol. 21, pp. 45–54, July 1987.

[126] N. Brière and P. Poulin, "Adaptive representation of specular light," *Computers Graphics Forum*, vol. 20, no. 2, pp. 149–159, 2001.

[127] H. S. M. Coxeter, *Projective Geometry*. New York: Springer-Verlag, 1987.

[128] R. L. Burden and J. D. Faires, *Numerical Analysis*. Brooks Cole, 2004.

[129] T. M. MacRobert, *Spherical Harmonics*. Oxford: Pergamon Press, 1967.

[130] C. Müller, *Spherical Harmonics*. New York: Springer-Verlag, 1966.

[131] R. J. Renka, "Interpolation on the surface of a sphere," *ACM Transactions on Mathematical Software*, vol. 10, pp. 437–439, 1984.

[132] L. L. Schumaker and C. Traas, "Fitting scattered data on spherelike surfaces using tensor products of trigonometric and polynomial splines," *Numerical Mathematics*, vol. 60, pp. 133–144, 1991.

[133] E. W. Cheney, *Approximation Theory, Wavelets and Applications*, ch. Approximation and Interpolation on Spheres, pp. 47–53. Springer-Verlag, 1995.

[134] W. Freeden, M. Schreiner, and R. Franke, "A survey of spherical spline approximation," *Surveys on Mathematics for Industry*, vol. 10, no. 1, pp. 29–85, 1997.

[135] R. Ramamoorthi and P. Hanrahan, "Frequency space environment map rendering," *ACM Transactions on Graphics*, vol. 21, pp. 517–526, July 2002.

[136] R. Ramamoorthi and P. Hanrahan, "An efficient representation for irradiance environment maps," in *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pp. 497–500, Aug. 2001.

[137] J. Kautz, P.-P. Sloan, and J. Snyder, "Fast, arbitrary BRDF shading for low-frequency lighting using spherical harmonics," in *Rendering Techniques 2002: 13th Eurographics Workshop on Rendering*, pp. 291–296, June 2002.

[138] P.-P. Sloan, J. Kautz, and J. Snyder, "Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments," *ACM Transactions on Graphics*, vol. 21, pp. 527–536, July 2002.

[139] D. L. Ragozin, "Constructive polynomial approximation on spheres and projective spaces," *Transactions of the American Math Society*, vol. 162, pp. 157–170, 1971.

[140] P. Dierckx, "Algorithms for smoothing data on the sphere with tensor product splines," *Computing*, vol. 32, pp. 319–342, 1984.

[141] R. H. J. Gmelig-Meyling and P. Pfluger, "B-spline approximation of a closed surface," *Journal of Numerical Analysis*, vol. 7, pp. 73–96, 1987.

[142] M. J. D. Powell, *Numerical Analysis II: Wavelets, Subdivisions, and Radial Basis Functions*, ch. The Theory of Radial Basis Functions in 1990, pp. 105–210. Oxford University Press, 1992.

[143] N. Dyn and C. A. Micchelli, "Interpolation by sums of radial functions," *Numerical Mathematics*, vol. 58, pp. 1–9, 1990.

[144] D. Shepard, "A two-dimensional function for irregularly spaced data," in *Proceedings of the 23rd ACM National Conference*, pp. 517–524, 1968.

[145] R. Franke, "Scattered data interpolation: tests of some methods," *Mathematics of Computation*, vol. 38, pp. 181–200, 1982.

[146] W. A. Light and E. W. Cheney, "Interpolation by periodic radial basis functions," *Journal of Mathematical Analysis and Applications*, vol. 168, pp. 111–130, 1992.

[147] G. E. Fasshauer, "Hermite interpolation with radial basis functions on spheres," *Advances in Computational Mathematics*, vol. 10, pp. 81–96, 1999.

[148] M. Golitschek and W. A. Light, "Interpolation by polynomials and radial basis functions on spheres," *Constructive Approximation*, vol. 17, pp. 1–18, 2001.

[149] U. Trottenberg, C. W. Oosterlee, and A. Schuller, *Multigrid*. Academic Press, 2001.

[150] S. Daly, *Digital Image and Human Vision*, ch. The Visible Differences Predictor: An algorithm for the assessment of image fidelity, pp. 179–206. MIT Press, 1993.

[151] V. Volevich, K. Myszkowski, A. Khodulev, and E. A. Kopylov, "Using the visual differences predictor to improve performance of progressive global illumination computations," *ACM Transactions on Graphics*, vol. 19, pp. 122–161, Apr. 2000.

[152] F. Drago and K. Myszkowski, "Validation proposal for global illumination and rendering techniques," *Computers & Graphics*, vol. 25, pp. 511–518, June 2001.

[153] M. R. Bolin and G. W. Meyer, "A visual difference metric for realistic image synthesis," in *Proceedings of SPIE 3644: Human Vision and Electronic Imaging IV*, pp. 106–120, 1999.

[154] F. C. Crow, "Shadow algorithms for computer graphics," in *Computer Graphics (Proceedings of SIGGRAPH 77)*, vol. 11, pp. 242–248, July 1977.

[155] M. McGuire, *GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics*, ch. Efficient Shadow Volume Rendering, p. 816. Addison-Wesley Professional, 2004.

[156] T. Lokovic and E. Veach, "Deep shadow maps," in *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, pp. 385–392, July 2000.

[157] M. Bunnell and F. Pellacini, *GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics*, ch. Shadow Map Antialiasing, p. 816. Addison-Wesley Professional, 2004.