SUPERPOSITION RENDERING:

Increased Realism for

Interactive Walkthroughs

by

Rui M. R. de Bastos

A dissertation submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill

1999

Approved by:

Advisor: Frederick P. Brooks, Jr.

Reader: Anselmo Lastra

Reader: Wolfgang Stuerzlinger

ABSTRACT

Rui M. R. de Bastos

SUPERPOSITION RENDERING: Increased Realism for Interactive Walkthroughs

(Under the direction of Dr. Frederick P. Brooks, Jr.)

The light transport equation, conventionally known as the rendering equation in a slightly different form, is an implicit integral equation, which represents the interactions of light with matter and the distribution of light in a scene. This research describes a signals-and-systems approach to light transport and casts the light transport equation in terms of convolution. Additionally, the light transport problem is linearly decomposed into simpler problems with simpler solutions, which are then recombined to approximate the full solution. The central goal is to provide interactive photorealistic rendering of virtual environments.

We show how the light transport problem can be cast in terms of signals-and-systems. The light is the signal and the materials are the systems. The outgoing light from a light transfer at a surface point is given by convolving the incoming light with the material's impulse response (the material's BRDF/BTDF). Even though the theoretical approach is presented in directional-space, we present an approximation in screen-space, which enables the exploitation of graphics hardware convolution for approximating the light transport equation.

The convolution approach to light transport is not enough to fully solve the light transport problem at interactive rates with current machines. We decompose the light transport problem into simpler problems. The decomposition of the light transport problem is based on distinct characteristics of different parts of the problem: the ideally diffuse, the ideally specular, and the glossy transfers. A technique for interactive rendering of each of these components is presented as well a technique for superposing the independent components in a multipass manner in real time.

Given the extensive use of the superposition principle in this research, we name our approach *superposition rendering* to distinguish it from other standard hardware-aided multipass rendering approaches.

To Claudine and Allan

ACKNOWLEDGEMENTS

With a “little” help of some friends...

- Thanks to Andrew Glassner for pointing out UNC-CH as a great school for graduate studies in computer graphics and to the Spring 1995 admissions committee for letting me in.
- Thanks to my PhD committee, for requesting “a more formal approach”.
- Very special thanks to my advisor, Dr. Fred Brooks, for ALWAYS understanding my ups and downs, and making me believe that I could make it. Thanks also for all the great opportunities you opened up for me, Fred.
- Special thanks to Anselmo Lastra, my “co-advisor”, friend, and member of my committee, for the uncountable discussions and readings of my manuscripts.
- Special thanks also to Mary Whitton, not in my PhD committee, but always present in my way through graduate school.
- Thanks to the other members of my PhD committee: Gary Bishop, Dinesh Manocha, Wolfgang Stuerzlinger, and Turner Whitted. Thanks in special to Wolfgang for brainstorms along the PhD and careful reading of this dissertation.
- Thanks to my friends Chris Wynn, Kenny Hoff, and Paul Zimmons, for great help with implementation of some techniques presented in this work. I must double acknowledge Paul Zimmons for the comments on early drafts of this dissertation. chapters.
- Thanks to the many members of the UNC Walkthrough team along the years 1995 to 1999, in special to those of the Brooks House and Hybrid Reality sub-teams.
- The entire Department for always providing an enjoyable place to work.
- Lightscape Technologies and Discreet Logic for a wonderful Summer of 1998. In special, Filippo Tampieri for all the challenges and excitment about my work.
- My wife, Claudine, for her continuous and unconditional love, and for being present even in the toughest times.
- My son, Allan, for all the joy he brought to my life.
- My family, for the endless encouragement.

This work was partially supported by CNPq/Brazil—201142/93-7, PRAXIS XXI—BD/16066/98, SGI, NIH/National Center for Research Resources—RR02170, NSF—MIP-9612643, and DARPA—E278.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

API — application programming interface

BDF — bidirectional distribution function (either BRDF or BTDF)

BRDF — bidirectional reflectance distribution function

BTDF — bidirectional transmittance distribution function

COP — center of projection

CRT — cathode-ray-tube

FOV — field of view

FPS — frames per second

HMD — head-mounted display

LASER — light amplification by stimulated emission of radiation

CHAPTER 1

INTRODUCTION

Synthesis of realistic images has been a major focus of research in computer graphics. The goal is to create a visual experience identical to that which an observer would have when looking at a real, existent or nonexistent, environment. To provide such convincing simulation, the intricate geometry and the physically complex lighting effects of the environment need to be reproduced—*photorealism*. In addition, images need to be generated at interactive frame-rates, so that the viewer can also perceive smooth motion in the scene—*motion realism*. The combination of photorealism and motion realism we call *realism* (Figure 1.1).

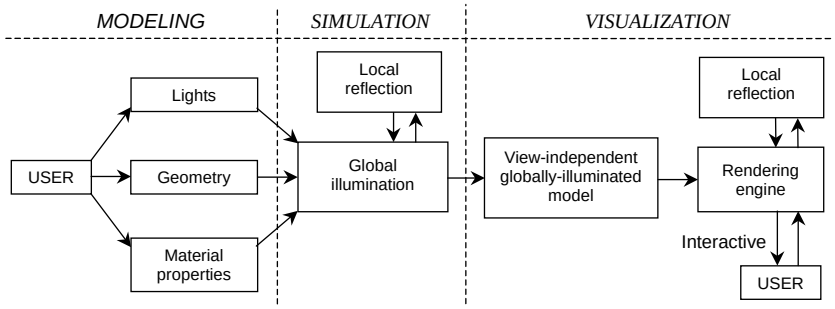


Figure 1.1: *Model-driven approach to realistic rendering—besides the model, five additional elements are fundamental for realistic image synthesis: a simulation of global light transport in the environment, sampling strategies for creating a view-independent globally-illuminated model, reconstruction techniques, a local reflection model, and an interactive rendering engine.*

Image synthesis research spans a spectrum (Figure 1.2). At one extreme, images are rendered for real-time viewing, no matter how unrealistic they look. At the other extreme, algorithms compute accurate images, no matter how long it takes to compute them. We want to combine advantages of these two and produce, at interactive rates, images that look right. The requirement for image accuracy

implies simulating light transport in the scenes, whereas the real-time requirement demands exploiting graphics hardware to improve image synthesis performance.

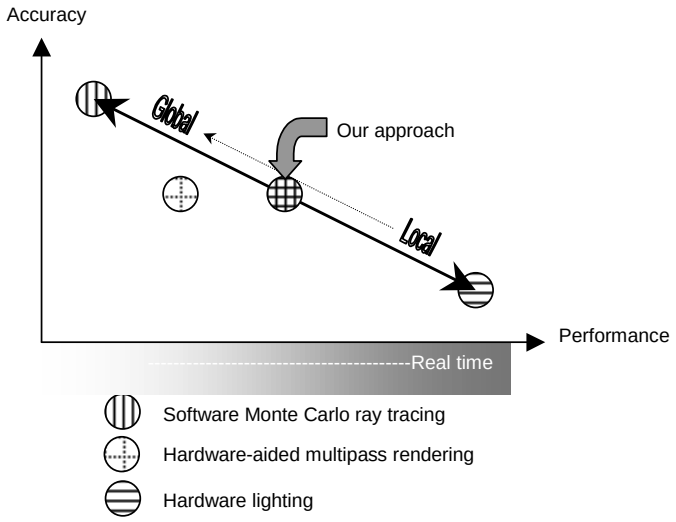


Figure 1.2: *Image synthesis spectrum.*

The ability to generate photorealistic images at interactive rates has important applications in virtual prototyping of building designs, automobile styling, and stage and set lighting design. For all of these, the demands for greater photorealism and higher frame-rates are always increasing. Pushed by this need for interactive synthesis of photorealistic images, graphics hardware systems implement fast rendering capabilities. Current systems can handle up to hundreds of thousands of texture-mapped, lighted, and anti-aliased triangles at interactive frame-rates.

The illumination models supported in hardware are not enough to provide the level of photorealism demanded by the visual experience goal. The hardware approximates only simple local-illumination models, whereas global-illumination models are necessary for simulating lighting details observed in nature such as shadows, reflections, translucency, refraction, color bleeding, diffraction, etc.

Local or *direct illumination* refers to the distribution of reflected light as a function of incoming energy directly from light sources without considering occlusion by objects along the light path. *Local* here defines surface illumination to depend only on the surface material properties and the characteristics of the light sources. There is no notion of the environment as a whole (Figure 1.3 (left)—notice that local illumination does not take into account occlusion of the table between the light source and the point on the floor).

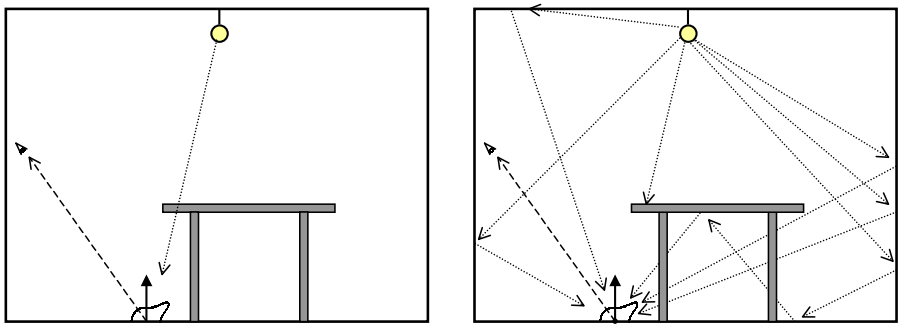


Figure 1.3: *Light transfers to compute local-illumination (left) and global-illumination (right) at a point on the floor.*

Global or *indirect illumination* takes into account the entire environment when computing illumination for each surface point. For example, shadows depend on determining occlusion between light sources and surface regions; reflections and color bleeding depend on the multiple interreflections of light, each weighted by the corresponding material properties (Figure 1.3 (right)). In reality, surfaces receive light both directly from light sources and indirectly from interreflections and transmissions of the environment. Global illumination can clearly produce more accurate approximations of photorealistic images than can the local-illumination models.

Both local- and global-illumination models approximate, to some extent, a steady-state solution of the *light transport equation*—an integral equation that describes the total light L_o leaving a surface point $\mathbf{x}$ in a given direction $\vec{\omega_o}$:

$$L_o(\mathbf{x}, \vec{\omega_o}) = L_e(\mathbf{x}, \vec{\omega_o}) + \int_\Omega f_{bd}(\mathbf{x}, \vec{\omega_o}, \vec{\omega_i})\ L_i(\mathbf{x}, \vec{\omega_i})\ cos\ \theta_i\ d\vec{\omega_i} \tag{1.1}$$

where L_e represents the light directly emitted by point $\mathbf{x}$ in direction $\vec{\omega_o}$ and the integral term represents the light indirectly transferred through point $\mathbf{x}$ into direction $\vec{\omega_o}$. The indirect term integrates incoming light L_i from all possible directions in the sphere Ω around the point in question $\mathbf{x}$, weighted by the corresponding directional reflectance/transmittance properties of the surface material f_{bd}. The key challenges in solving the light transport equation are:

- the integration of incoming light over the entire directional domain—the sphere of directions—centered at each visible point in a scene and
- the recursiveness of the problem implied by the implicit form of the equation—the incoming light L_i with respect to the point in question is *per se* the recursive application of the same equation at the points in the scene where L_i originates.

Note how naively local illumination tries to approximate the light transport equation by considering that light reaches a point originating only from light sources in the scene—there is no notion of environment and there is no notion of interreflections (recursion) either. In the simplest, and quite common, approximation, the global term of the light transport equation is approximated with a constant additive value that tries to model the ambient light in a scene [Foley90].

There exist approaches for approximating the light transport equation accurately. Glassner [Glassner95] discusses several methods for solving integral equations such as the light transport equation. The methods range from symbolic solutions to numerical and Monte Carlo integration. However, those methods cannot be evaluated by software fast enough to satisfy the real-time requirement. Moreover, none of them is simple enough to justify implementing in hardware, which would approach real-time performance.

General global-illumination research in Monte Carlo ray tracing [Veach98, Lafortune96, Jensen96, Ward94] properly accounts for all the components of illumination. Most such research focuses on an image-driven approach—the creation of a single accurate image in a reasonable amount of time (ranging from seconds to hours of processing time on a single 190 MHz MIPS R10000 processor [Veach98]). Emphasis is placed on high quality, measurable accuracy, and numerical robustness of the method. The computational time for obtaining reasonably converged global-illumination for a single image with such techniques prohibits interactivity. Recently, there has been some interest in using ray-tracing algorithms in multiprocessor machines [Parker99, Parker98]. The approaches exploit, in a "brute-force" manner, conventional, and expensive, shared-memory multiprocessor machines to produce interactive ray-traced images for low complexity models by using dozens of processors—a model of $75,000$ polygons and a spline teapot renders at 4 frames per second for images of 512×512 pixels on 60 processors of an SGI^{TM} Origin 2000 workstation [Parker99]. Ray tracing could benefit from custom hardware. However, even after more than twenty years of ray-tracing research, custom graphics hardware compatible with such techniques does not yet seem on the horizon.

Alternatively, standard hardware-aided multipass rendering [Diefenbach96, McReynolds98] uses features of most current inexpensive graphics hardware to approximate global illumination by computing at each frame (view pose) the multiple lighting effects observable in nature. This type of approach requires rendering the entire scene multiple times per frame, i.e., handling the entire complexity of the scene several times per frame. For realistic model complexities, this cannot now be done at interactive frame rates—computational time is in the order of seconds for scenes with tens of

thousands of triangles on SGI^{TM} RealityEngine workstations [Diefenbach96]. The scenes have to be rendered multiple times, which easily overwhelms even the best graphics architecture. For example, mirror reflections on planar surfaces are computed by mirroring the view pose with respect to each planar reflector and rendering the scene from each reflected view pose. Glossy reflections are handled with a jitter-and-average approach; the reflected view pose is jittered several times with respect to each reflector and the resulting rendered mirror reflections are accumulated and averaged to produce a glossy looking reflection. View-independent lighting and shadows are also approximated with additional passes over the entire scene database.

Neither Monte Carlo ray tracing nor standard hardware-aided multipass rendering can yet provide interactive rendering of photorealistic images for reasonably complex models. Visibility computation and mathematical integration of incoming light is the bottleneck. Ray tracing computes visibility by evaluating the intersection of pixel rays with primitives in the scene, whereas hardware-aided multipass rendering uses a depth buffer to perform the primitive sorting along pixel directions. In terms of mathematical integration of incoming light, ray tracing integrates by summing contributions from several reflected/refracted rays in a solid angle (Figure 1.4(left)), whereas hardware-aided multipass rendering integrates by summing jittered contributions that map to the same pixel in an image (Figure 1.4(right)).

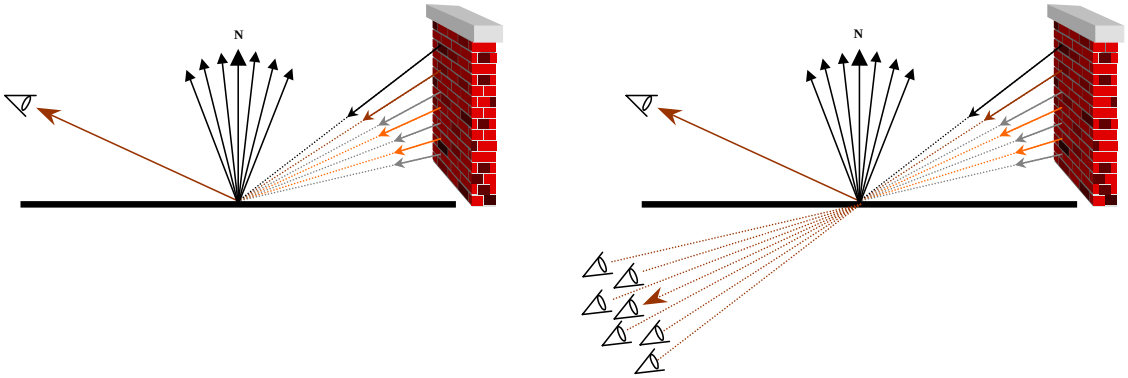


Figure 1.4: *Monte Carlo ray-tracing (left) and jitter-and-average (right) for a glossy light tranfer at a point on a floor. Notice the distribution of normals at the point in question on the floor that produces the distribution of reflected rays for Monte Carlo ray tracing and the jittered reflected cameras for multipass rendering.*

Let's consider the data access patterns of standard hardware-aided multipass rendering and of standard Monte Carlo ray tracing (see Figure 1.4). Multipass rendering has better data locality than ray tracing. In multipass rendering, the whole scene is sequentially rendered several times per frame

depending on the number of rendering passes required by the multipass approach. In Monte Carlo ray tracing, a spatial search over the scene to determine the first visible primitive has to be done for each pixel per frame. The number of traversals of the data set (scene) is potentially smaller for the multipass approach than it is for the ray-tracing approach, since there are more pixels on the screen than passes in the multipass approach.

Consider also how the two approaches traverse the data set. Multipass rendering is an object-order method, whereas Monte Carlo ray tracing is an image-order approach. The atomic primitive in hardware-aided multipass rendering is a polygon, whereas in ray tracing the atomic primitive is a ray. The multipass rendering approach accesses a single independent polygon for an atomic operation (very local information), whereas the ray-tracer has, potentially, to access the entire scene for each atomic operation (very global information). Potentially, a machine performing an atomic operation in ray tracing has to access more data than in multipass rendering.

In summary, the visibility and the integration techniques of Monte Carlo ray tracing and multipass rendering imply distinct memory access patterns and distinct computational times. In order to provide more interactive frame rates, our method, besides exploiting the graphics hardware, also improves on handling visibility information and on mathematical integration.

We present a mathematical framework for solving the light transport equation by using hemispherical convolution. To make the solution of the light transport equation practical, we also present a linear decomposition of the light transport equation, methods for approximating each of the components, and techniques for their recombination. These methods exploit linear superposition, spatial coherence, and spatial locality of the data. Computers are good at performing linear operations and this dissertation exploits the linear superposition principle at two distinct levels. At the highest level, the full light transport problem is broken down into smaller problems with simpler solutions, which are then recombined (by superposition) to approximate the full solution. The light transport equation is linearly decomposed based on the nature of the light transport problem and on the linear operators available in current graphics hardware. From that division of the problem we precompute as much as possible of the light transport components and recombine them at rendering time to achieve interactivity. This approach allows exploiting current graphics hardware in a runtime, multipass manner for the recombination of the linearly independent components. At a lower level, one of the components, namely the glossy component, exploits hardware-assisted convolution to provide image-space integration of pixel neighborhoods. Both the decomposition and the runtime rendering

exploit coherence in reflected/transmitted space to reduce visibility computations for glossy light transfers.

This approach is much faster than standard hardware-aided multipass rendering because of

1. the reduction of data to be rendered per frame (fewer number of rendering passes over the whole model database) due to precomputation of view-independent quantities, which are computed at runtime for standard multipass rendering—ideally diffuse transfers, shadows, and visibility for non-diffuse reflections;

2. visibility assumptions, because we assume that visibility for glossy reflections is captured with a single mirror-like reflection, instead of requiring multiple jittered images; and

3. per pixel hardware-aided mathematical integration through image-space convolution, instead of independent weighting and accumulating for each jittered image.

Since substantial preprocessing is involved, our full approach for approximating the light transport equation applies only to static scenes—only the viewer is allowed to move; light sources and objects remain static—even though, parts of the method are applicable to dynamic scenes. The proposed methods are also simple enough for hardware implementation and, if so implemented, will make per-pixel approximations of the light transport equation rapid for arbitrary models in arbitrary representations.

1.1 Overview of Our Signals-and-Systems Approach to Rendering

This section summarizes the approach developed in this dissertation for achieving interactive rendering of photorealistic images. We start by discussing light transfers related to a single point in an environment. Then we move on to the combination of light transfers to form a light path, and then on to the full global illumination problem. The illumination problem is treated in terms of *signals-and-systems*. The concepts of *signal*, *system*, *impulse response*, and *convolution* are necessary for understanding this discussion. The reader unfamiliar with these concepts is referred to the brief overview in Chapter 2 or to [Oppenheim96].

The full light transport or global-illumination problem is described by the light transport Equation (1.1). The outgoing light from a point in a scene is the sum of a local term and a global term. The local term represents the light emitted by the point in question. The global

term describes the integration of all the light reaching the point in question from somewhere else in the scene, and being transferred back into the environment through reflection or transmission at the point in question, weighted by the corresponding material properties. The bidirectional reflectance/transmittance distribution functions (BRDF/BTDF) represent how the behavior of light transfers depends on material properties. Light transfers such as emission, reflection, and transmission exhibit a variety of behaviors. *Diffuse* transfers correspond to an equal scattering of light in all directions. *Specular* transfers scatter incoming directional light in a narrow band of reflected directions (ideal specularity means that there is only one direction in which outgoing light may go). Non-ideal or *glossy* transfers describe the continuum between ideally diffuse and ideally specular transfers. Glossy transfers have a directional bias but are neither restricted to a single outgoing direction nor effect an equal scattering to the whole outgoing hemisphere.

A light transport path starts at a light source and ends at a sensor. The sensor is usually a viewer. We abstract away the light creation process and the perceptual process produced by light. We consider only what happens with light in-between, i.e., from the time light leaves the emitter until it reaches the viewer's eye. A light path starts at a point on the surface of a light source and leaves in a particular direction. Assume that the environment is composed by media of uniform material separated by surface interfaces. The light leaving a point on a source in a given direction travels rectilinearly and undiminished until it reaches another point on the surface of a light receiver. At that interface, three phenomena can occur as a response of the system to incoming light:

- the light can be absorbed without a resulting signal in the same space, i.e, no outgoing light;
- the light can be reflected back into the original medium producing a modified signal—light in a new direction;
- or the light can be transmitted into the new medium producing a modified signal—light in a new medium and in a new direction.

In terms of signals-and-systems, the incoming light is the input signal, the receiver point is the system, and the outgoing light is the output signal. The function describing the behavior of the outgoing light is the point spread function[1]—the impulse response—of the system. Note the relative difference of the

[1] In optical terms, a *point spread function* is the image produced by a point light source through an optical system. In our approach, the point spread function describes the spread of light from a single ray of light incident on a point at a given orientation, i.e., the usual bidirectional reflectance/transmittance distribution function of materials—the ratio of outgoing to incoming light in the directional space around the point in question.

incoming and outgoing terms with respect to receiving and emitting surface points; the outgoing light from an emitter corresponds to the incoming light of a receiver.

From signal processing theory, the output of a system is given by the convolution of the input signal with the system's impulse response. From our analogy of the light transfer problem with signals and systems, this implies that the outgoing light from a light transfer is the result of convolving the incoming light with the material BRDF/BTDF at the point in question. Formally, compare the convolution equation

$$g(x) = f(x) \star k(x) = \int_{-\infty}^{+\infty} f(u)\ k(x-u)\, du. \tag{1.2}$$

with the second term of the light transport Equation (1.1). Both integrate the product of two functions. This suggests that the light transport equation can be expressed in a simpler form using convolution:

$$L_o(\mathbf{x}, \vec{\omega_o}) = L_e(\mathbf{x}, \vec{\omega_o}) + L_i(\mathbf{x}, \vec{\omega_i}) \star K(\mathbf{x}, \vec{\omega_{io}}, \vec{\omega_i}) \tag{1.3}$$

where $K(\mathbf{x}, \vec{\omega_{io}}, \vec{\omega_i})$ is the convolution kernel derived from material bidirectional distribution functions, as will be discussed in Chapter 4.

There is a necessary mental shift for understanding this new representation for the light transport equation in terms of convolution. The reader should note that a convolution kernel is a function defined in the same space as its corresponding signal. Instead of defining convolution in the traditional rectilinear space with an infinite range, as in Equation (1.2), the light transfer problem is carried out in a directional space around the point of interest. Both the signal and the kernel are defined in this directional space in terms of directions (or angles, in spherical coordinates).

The global-illumination problem also involves the recursive application of the light transport equation. The illumination at a point on a surface is determined by taking into account all the light received from the rest of the environment into that point; however, points on the rest of the environment also depend on the application of the same equation. A recursive algorithm is necessary to solve the full light transport problem. A single application of the light transport Equation (1.3) represents a light transfer at a point $\mathbf{x}$, whereas multiple recursive applications correspond to light paths.

Theoretically, casting the light transport equation in terms of convolution does not represent a considerable advantage over its traditional integral form. However, in practice, convolution is readily available in some graphics architectures, which represents an opportunity for hardware-aided evaluation of the light transport equation. In addition, we do not want to apply the convolution-based approach for all the types of light transfers. Ideally specular transfers do not need convolution

at all, since the corresponding convolution kernel would be a delta function in directional space. Also, the ideally diffuse transfers can be precomputed using the well-known radiosity method. We propose the application of the convolution approach for glossy transfers, which are in general hard to simulate with other approaches. Moreover, our convolution-based approach assumes that visibility for glossy reflections is captured with a single mirror-like reflection on the glossy surface, which dramatically reduces visibility evaluation—reduces the number of rendering passes to one if compared to the many passes of standard hardware-aided multipass rendering for approximating glossy reflections. Additionally, mathematical integration of light is performed per pixel by using image-space convolution, instead of weighting and accumulating several rendering passes of approximately the same (jittered) scene in standard multipass rendering.

1.2 Thesis

This dissertation presents a multipass rendering method for the interactive approximation of global illumination in static environments based on the linear decomposition of the light transport equation and its runtime recombination through the use of current graphics hardware. Some of the underlying components are precomputed based on view-independent quantities and on their suitability for hardware processing. Briefly, we propose to synthesize photorealistic images of diffuse and non-diffuse static environments at interactive frame-rates with current graphics hardware.

The thesis of this research is:

> *Global illumination techniques can be used for interactive photorealistic walkthroughs of non-diffuse virtual environments. By linearly decomposing the light transport problem and by exploiting graphics hardware, multipass rendering and image-space convolution can provide interactive performance and natural appearance for walkthroughs of static, globally illuminated, non-diffuse environments.*

The synthesis of photorealistic images at interactive frame-rates demands the solution of four main problems:

1. How can one use the decomposition of the light transport equation to achieve performance improvements on the rendering of globally-illuminated scenes? Chapter 4 decomposes the light transport equation into linear components that are suitable for preprocessing and for exploiting current graphics hardware. Then, the succeeding chapters discuss each of the components.

2. How should one preprocess and represent the view-independent component of global illumination? Chapter 5 is devoted to the processing of radiosity data to represent view-independent illumination efficiently and to render this component using the graphics hardware efficiently.

3. How can one render ideally specular light transfers at interactive frame-rates? Chapter 6 presents an image-based alternative for computing mirror-like reflections. The approach extends to ideal transmission.

4. How can one render glossy reflections in real time? Chapter 7 exploits spatial coherence in reflected images to approximate glossy reflections using hardware-aided image-space convolution. The approach extends to translucency.

This dissertation focuses on interactive walkthroughs that account for view-independent and view-dependent illumination. Going beyond previous global-illumination research, we emphasize structuring the scene database to allow for fast rendering with view-dependent illumination at interactive rates from arbitrary viewpoints. In addition to capturing purely specular view-dependent effects, the approach in this dissertation properly captures the imperfect reflection scattering of glossy surfaces. Due to the light scattering of glossy reflections, higher-order reflections often contribute little to the final scene's illumination. The approach in this dissertation is restricted to glossy, first-order, planar reflections; however, the Future Work section discusses how to extend the techniques to recursive reflections and to curved surfaces.

Given the extensive use of the superposition principle in this research, we name our approach *superposition rendering*, to distinguish it from standard hardware-aided multipass rendering, which also exploits superposition in a slightly different way. The main distinctions between standard hardware-aided multipass rendering and superposition rendering are:

Hardware-aided multipass rendering	*Superposition rendering*
Multipass shadows	Single pass precomputed radiosity shadows
Multipass jitter-and-average glossy effect	Single pass and convolve
Phong local lighting	Reflectance mapping on mirror reflections

Table 1.1: *Feature comparison of hardware-aided multipass rendering and superposition rendering.*

In summary, in terms of performance, the major advantage of superposition rendering is due to a dramatic reduction in the number of rendering passes required per frame to approximate the light transport equation. For example, whereas multipass rendering may typically take tens of passes on

the entire scene to produce a glossy reflection, superposition rendering typically takes a single pass on the entire scene and a convolution pass in image space.

1.3 Results

We present superposition rendering—a hybrid geometry- and image-based multipass rendering approach that offers the following:

- a decomposition of the global-illumination equation into linear components that exploits current graphics hardware for its efficient recombination,
- techniques for reconstruction of view-independent illumination and its resampling for storage as textures with improved smoothness, exploiting current graphics hardware,
- a sampling-and-reconstruction scheme for preprocessing visibility information for non-diffuse light transfers,
- the use of convolution for approximating view-dependent solid angle light integration in glossy reflections to simulate material properties (both object-space and image-space approaches are presented), and
- a technique for approximating the view-dependent reflectance/transmittance kernel as a view-independent convolution kernel texture (for integration purposes) and a view-dependent reflectance texture (for directional modulation) for certain types of BRDFs.

Figure 1.5 sketches the flow of data in superposition rendering. The preprocessing phase appears on the left and the runtime phase is on the right. A simple scene is used for illustrating the method; the pyramid is made of glossy copper material, and the other objects are all made of ideally diffuse materials. As part of the preprocessing, we compute a radiosity solution for the scene and generate a set of images-with-depth for each non-diffuse reflector in the scene. Also as part of the preprocessing phase, we compute a kernel texture and a reflectance sphere map for each non-diffuse material. Then, at runtime, given a view pose, a subset of the images-with-depth is reprojected to reconstruct the mirror-reflected image on each non-diffuse reflector. After the mirror-reflected image has been computed, it is convolved with the BRDF-based kernel of the corresponding material. After the reflected image is blurred, it is also view-dependently modulated. Finally, using the same view pose, the view-independent component is rendered and added to the specular component already

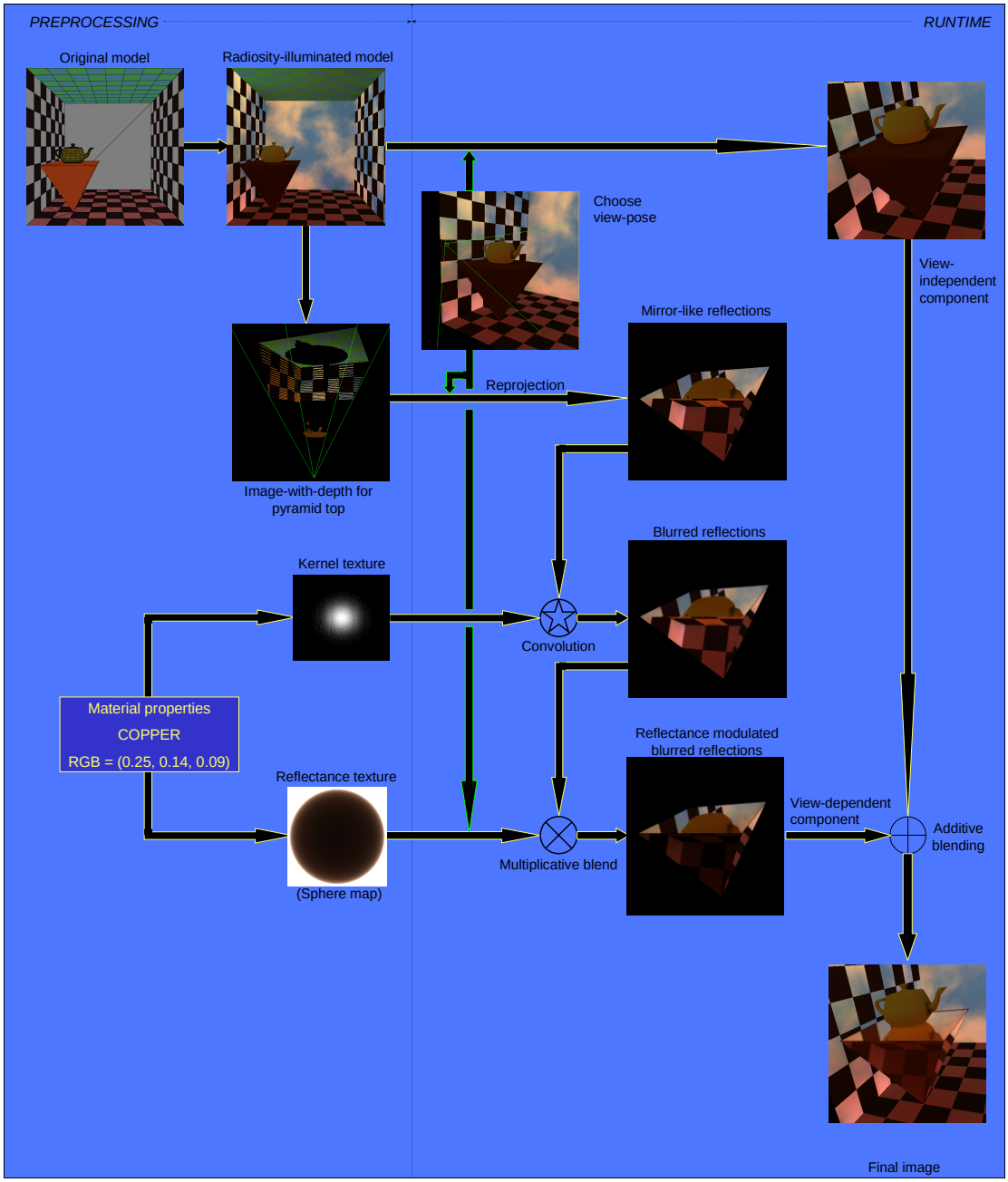


Figure 1.5: *Superposition rendering of a simple scene. The pyramid is made of glossy copper material. All the other objects are ideally diffuse.*

in the frame-buffer. Both the multiplications and additions in the multipass rendering method are implemented with blending functions available in current graphics hardware systems.

1.3.1 View-Independent Component

The view-independent component will be discussed in Chapter 5. We use the radiosity method for precomputing and rendering the view-independent component of the light transport equation. The radiosity method works in two stages—a sampling stage and a reconstruction stage. The sampling stage is performed offline as a preprocessing of the scene—the initial polygons representing the geometry of the scene are adaptively subdivided and radiosity is computed at the vertices of the subdivided primitives. The reconstruction stage is performed at runtime by shading the polygons of the scene with the corresponding radiosity information.

Traditionally, radiosity results are rendered using the dense adaptively subdivided meshes of polygons with per-vertex radiosity. Clearly, this mesh-based rendering approach is *transformation bound*—there are potentially more primitives to be geometrically transformed than the graphics hardware can handle at interactive rates. Our approach, similarly to what other researchers did, converts the densely tesselated meshes into textures and renders radiosity information by painting the unsubdivided polygons of the scene with radiosity textures. This texture-based approach trades performance for texture memory. The radiosity textures are preloaded into texture memory and queried for shading the unsubdivided polygons of the scene. Since current graphics architectures contain only up to 64 megabytes of texture memory, in general, only a fraction of the subdivided polygons in a scene is converted to textures—the polygons with higher subdivision rates.

Since the standard polygon shading technique in hardware is bilinear interpolation for both the per-vertex scheme and the texture-based scheme, the final images may present artifacts due to slope discontinuities of the shading across the edges between adjacent polygons. We have experimented with higher-order filtering schemes both as runtime filtering and as prefiltering techniques. The runtime filtering technique provides smoother results without requiring any modification of the radiosity data, but demands using a slower and non-standard bicubic texture-filtering mode. The prefiltering scheme also provides smoother results by prefiltering the radiosity data and by using the standard (and faster) bilinear texture-filtering mode.

In terms of the graphics hardware, the rendering of the view-independent component benefits from standard polygon rendering, from regular texture mapping and filtering, and from texture memory size and access time.

1.3.2 The Non-Scattered Transfers

The non-scattered transfers of light will be discussed in Chapter 6. We discuss both a geometry-based approach and an image-based approach for rendering mirror-reflected images on planar surfaces. The geometry-based approach re-renders the entire scene for each mirror-reflected view pose. The image-based approach precomputes images-with-depth for a set of mirror-reflected view poses with respect to each planar mirror in the scene and selects, at runtime, the image or set of images that best approximate the current mirror-reflected view pose. Both the geometry-based and the image-based approaches can be used for computing mirror-reflected images in superposition rendering.

The geometry-based approach does not introduce any artifact on the mirror-reflected images, whereas the image-based approach introduces disocclusion artifacts and noise due to the presampling and reconstruction stages. However, the image-based approach is potentially faster than the geometry-based approach, since the image-based approach does not have to re-render the entire scene for each mirror-reflected image. Both the geometry-based and image-based approaches use the stencil buffer to restrict the rendering of mirror-reflected images to the projected areas of the reflective surface on the screen [McReynolds98].

In terms of the graphics hardware, the rendering of the view-dependent non-scattered component benefits from standard polygon and point rendering and pushes the geometrical transformation capabilities of the hardware.

1.3.3 The Directionally-Dependent Non-Scattered Transfers

The directionally-dependent non-scattered transfers will be discussed in Chapter 6. Most polished materials exhibit higher reflectance values at grazing angles than at normal incidence and may also filter the color of reflected light in a directionally dependent manner—*mirror-like* materials. These directional-dependent reflectance variation and color filtering effects are approximated by processing a mirror-reflected image computed on the reflective surface.

Initially, a mirror-reflected image is computed as described in the previous section for the screen region where the reflective surface projects. Then, the reflective surface is rendered using a

multiplicative blending function for modulating the reflected image. Constant color modulation of the reflected image is achieved by using a constant color or texture mapping of the reflective surface. Directionally dependent reflectance is produced by using view-dependent texture mapping such as sphere mapping. The sphere maps representing the directional-dependent reflectance are precomputed and attached to the reflective surfaces.

In terms of the graphics hardware, the rendering of directionally-dependent non-scattered reflections depends on the same parts of the graphics hardware as the rendering of non-scattered reflections, but also depends on view-dependent texture mapping (e.g. sphere mapping).

1.3.4 The Scattered Transfers

The scattered transfers of light will be discussed in Chapter 7. We present both an object-space convolution approach and an image-space convolution approach. The image-space approach is also split into a spatially invariant convolution approach and a spatially variant convolution approach. From the three approaches presented, only the image-space spatially invariant convolution approach can efficiently benefit from graphics hardware to provide glossy reflections at interactive rates.

The image-space spatially invariant convolution approach to glossy reflections starts by computing a mirror-reflected image on the planar surface, which is then blurred and view-dependently modulated as described in the previous sections. Both the convolution kernel and the view-dependent modulation factors are derived from the BRDF of the glossy material.

In terms of the graphics hardware, the rendering of scattered reflections benefits from an implementation of image-space convolution and from view-dependent texture mapping (sphere mapping). The image-convolution technique pushes the screen fill capabilities of the hardware.

1.4 Images and Numbers

Figures 1.6 to 1.8 show the results of superposition rendering for a room of a six-room house model with $140,000$ polygons and five planar glossy reflectors. We analyzed the performance of superposition rendering with that house model. Images were rendered at 720×486 pixels and performance data were collected by playing a pre-recorded path (944 frames) through the house and using two images-with-depth per hemisphere for each glossy reflector in the scene. Performance was measured with superposition rendering on an SGI ^{TM} Onyx2 workstation using a single 250 MHz R10000 processor and a single InfiniteReality2 graphics pipe with four raster managers. The average frame-rate

for that model ranged from 30 fps to 8 fps by varying the number of glossy reflectors in the scene from zero to five (see Chapter 6 for more detailed results).

Figure 1.6: *Superposition rendering using geometry-based reflections.*

Figure 1.7: *Superposition rendering using image-based reflections (two images-with-depth per hemisphere for each reflector).*

We have also analyzed the performance of the individual operations involved in our runtime rendering. We ran a series of tests on the house model using our superposition rendering and selecting two images-with-depth for each hemisphere in the image-based reflections. Allowing the number of reflectors to vary, we found that time spent rendering the view-independent component of the scene was invariant with respect to the number of reflectors. We also found that the convolution and modulation operations combined for no more than 3% of the total rendering time. This suggests that the two operations do not dominate in the evaluation of our shading equation. It is important to note that the convolution operation is a screen-space operation; consequently, the cost is dependent on the screen-space projected area of each reflector and not on scene complexity.

Figure 1.8: *Superposition rendering using image-based reflections (two images-with-depth per hemisphere for each reflector) and illustrating the view-dependent variation of glossy reflections magnitude with orientation of glossy surfaces with respect to the viewer. Compare the magnitude of the glossy reflections on the floor in this figure and in the previous figure.*

In all these test runs with the house model, the image-based approach outperformed the geometry-based one, but this performance was not free. The image-based approach incurs additional memory overhead. The house model required 50 megabytes of storage of which 20% was for geometry-related data and 80% was for images-with-depth.

Figures 1.6 and 1.7 compare image quality of geometry-based reflections with our image-based approach. The image-based approach exhibits loss of detail and artifacts in some regions of the image. This is unfortunate for mirror-like reflections, but reasonable for glossy reflections. The primary causes for these artifacts are due to disocclusions and to noise introduced by the point-based reconstruction used by the image-based reflections.

Our BRDF decomposition into a view-independent convolution kernel and a view-dependent modulation for simulating glossy surfaces convincingly approximates the view dependent reflectance for glossy surfaces. Figures 1.7 and 1.8 show the results on the music room of the house model. In the first figure the view-direction is parallel to the floor and the piano top, while in the second one the viewer is looking down. Notice the increase in specular reflectance at grazing angles and the appropriately decreasing view-dependent component at higher angles.

Clearly, the blur obtained with convolution approximates the roughness of the surface at small neighborhoods. Unfortunately, the spatially invariant convolution approach assumes that reflected points are at a constant distance from the glossy reflector. A more accurate implementation performs spatially variant convolution based on depth of reflected points—points closer to the reflector get less blurred than points farther away. Chapter 7 describes a software implementation of spatially

variant convolution, which produces more photorealistic results than the spatially invariant illustrated in Figure 1.5 (see Figure 1.9—notice that points on the scene closer to the glossy surface are sharper than points farther away only for the spatially variant approach). However, such software implementation is not integrated into our multipass approach because of its low performance. We hope that next-generation graphics hardware will implement spatially variant convolution and allow the synthesis of images even more photorealistic using our multipass approach.

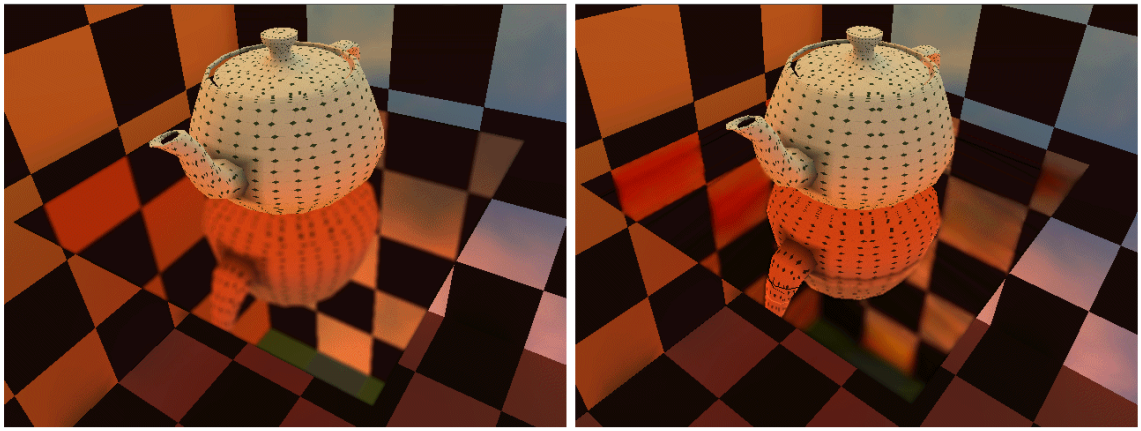


Figure 1.9: *Spatially invariant convolution (left) and spatially variant convolution (right) to approximate glossy reflection on planar surface.*

The sphere mapping-based technique for approximating directional reflectance is very effective. Besides providing a good approximation of directional reflectance, sphere mapping is available in graphics hardware from SGI^{TM}, which makes it a fast operation. However, sphere mapping suffers from both inherent and implementation problems. First, the mapping from viewing directions to texel coordinates in a sphere map is non-linear, which implies that directional resolution is not constant in the map. Secondly, as any other texture mapping technique, each vertex of a triangle or quadrilateral (primitive) is mapped onto a texture by computing texture coordinates. The edges of a primitive are mapped to straight lines connecting adjacent vertices of the primitive in texture space. This mapping of edges of the primitive to straight lines in the texture map is valid for rectangular textures, but incorrect for sphere maps—edges of a primitive should map to arcs in sphere mapping. Thirdly, connected vertices in a texture map are usually linearly interpolated for filling in the corresponding primitive. However, given the non-linearity of the mapping function, bilinear interpolation is not the appropriate interpolation for sphere mapping. Fourthly, related to the two previous problems, filtering is usually performed with a square filter in a rectangular texture. Sphere mapping should use a wedge-shaped filter defined in terms of the radius and theta coordinates of sphere mapping, instead. Additionally,

current implementations of sphere mapping in the hardware do not perform the expected wrapping of the sphere map texture. Consider a triangle seen at a grazing situation. The three vertices map to the outer region of a sphere map and the filled triangle should get texels only from the outer (grazing) region of the sphere map. However, the current implementation crosses the sphere map and may incorrectly provide texels that go through the non-grazing region. Finally, our particular use of sphere mapping technique for directional reflectance stresses the non-linearity problem of the technique. Our application requires high resolution on a region (outer ring) of sphere maps where the sphere-mapping technique offers low resolution. Though we reduced this problem by uniformly increasing the resolution of the whole map, a better solution would be to flip the sphere-mapping indexing scheme in the hardware. Grazing angles would index texels in the center of the sphere map, where there is higher resolution than in the ring.

CHAPTER 2

SIGNAL PROCESSING

The field of signal processing studies signals and systems, and their interaction. Signals and systems arise in several contexts and fields. A *signal* describes a phenomenon, whereas a *system* responds to an input signal to produce an output signal. For example, currents and voltages as a function of time in electrical circuits represent signals, whereas the circuits themselves represent systems. As another example, light represents a signal; and a photographic camera producing a projected image is a system. Digital images are also signals; and image-processing programs producing new images are systems. The field of signal processing defines a powerful mathematical framework for describing and analyzing signals and systems [Oppenheim96]. In this chapter, parts of this framework are presented. The intention is not to cover the whole signal-processing field, but to limit our description to the concepts relevant to understanding the topics developed in this dissertation.

Signal processing theory has been widely applied in computer graphics [Glassner95], [Watt92], [Wolberg92], [Foley90]. In general, the application has been related to aliasing artifacts and anti-aliasing techniques in digital images. Aliasing artifacts in discrete signals refer to high-frequency content being aliased to low-frequency information on the signals. In computer graphics, the digital images are multidimensional signals; and anti-aliasing techniques are the systems. Besides using this general application of signal processing theory to computer graphics, this dissertation introduces a new approach to solving the radiance equation—an integral equation—based on signal processing principles.

2.1 Signals and Systems

Both signals and systems can be defined in continuous domains and in discrete domains. Most of the basic signal and system properties and insights apply in both domains. A distinction will be made between the two whenever necessary.

2.1.1 Signals

Signals are represented mathematically as functions of one or more independent variables. For example, currents and voltages in electrical circuits are represented as functions of one dimension, time; pixel intensities in digital images are defined as functions of two dimensions, the spatial coordinates of the pixels. The independent variables thus depend on the context of the signals; they may represent time, space, or any other dimension where the signal is defined. However, signals of the same dimensionality have similar treatment with signal processing tools. For example, signals defined in a two-dimensional Euclidean space ($f(x, y)$) obey the same signal processing rules of a signal defined in two-dimensional polar coordinates ($f(r, \theta)$). The key element is the dimensionality of the space and not its shape. Also, both the independent and the dependent variables of a signal can be continuous or discrete. In this work, it is assumed that the state of the independent variable implies the same state on the dependent variable, i.e.; a continuous independent variable implies a continuous dependent variable and a discrete independent variable implies a discrete dependent variable. Thus, continuous signals are defined for a continuum of values of each of the independent and the dependent variables, whereas discrete signals are defined only for discrete sets of values of each of the independent and the dependent variables.

2.1.1.1 Impulse signal

An especially important signal is the impulse function, known also as the *Dirac delta function*. The *impulse function* is an infinitely narrow spike of infinite height whose area integrates to unity. It is zero everywhere but at zero on the domain, where it has an infinite value. Mathematically, the Dirac delta function is given by

$$\delta(x) = 0, \text{ if } x \neq 0$$

$$\int_{-\infty}^{+\infty} \delta(x)\, dx = 1. \tag{2.1}$$

Note that the nonzero point of the delta function can be moved by a shift operation. For example,

$$\delta(x - x_0) = 0, \text{ if } x \neq x_0 \tag{2.2}$$

is zero everywhere but at x_0.

The delta function has an interesting *inspection* or *sifting* property:

$$f(c) = \int_{-\infty}^{+\infty} f(x)\, \delta(x - c)\, dx \tag{2.3}$$

which isolates a single point on a curve. It represents the sampling of the continuous function $f(x)$ at the point c. The only point where the product in the integral above is not zero is at $x = c$, according to the shift property in Equation (2.2).

In the discrete domain, for integer n, the analogous function is the *Kronecker delta function* given by

$$\delta(n) = \begin{cases} 0 & \text{if } n \neq 0 \\ 1 & \text{if } n = 0 \end{cases}$$

which has similar properties to the Dirac delta function.

2.1.2 Systems

In signal processing, a *system* is anything that takes an input signal, $f(x)$, and produces an output signal, $g(x)$. A system is a "black box". Signal processing does not look into what is inside the system; it is concerned only with the relationship between the input and the output signals.

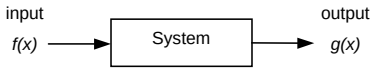


Figure 2.1: *A system: relationship of the output signal on the input signal*

Both continuous and discrete systems observe a number of basic properties. This discussion is limited to the properties that are relevant in our work. The interested reader is referred to [Oppenheim96] for a complete presentation of system properties in signal processing.

2.1.2.1 Linearity

Most of the signal processing theory is restricted to linear systems. *Linear systems* obey the *superposition principle*; that is, a weighted sum of several signals applied to a linear system produces

the weighted sum of the responses of the system to the individual signals. Formally, let input $f_1(x)$ produce output $g_1(x)$ and input $f_2(x)$ produce output $g_2(x)$ through a given system, i.e.:

$$f_1(x) \rightarrow g_1(x) \text{ and } f_2(x) \rightarrow g_2(x). \tag{2.4}$$

The linearity or superposition property requires that

$$a\, f_1(x) + b\, f_2(x) \rightarrow a\, g_1(x) + b\, g_2(x). \tag{2.5}$$

Many real-world systems can be well approximated by model systems that are linear. For example, electrical circuits made up of only resistances, capacitances, and inductances can be so represented. Image display systems, such as CRTs, typically have broad response regions that can also be so approximated.

2.1.2.2 Shift-invariance and shift-variance

Another important property of systems analyzed with signal processing is related to the shift of the input signal and its implications in the output signal. A *shift-invariant system* takes a shifted input signal and produces a shifted version of the output signal: the nature of the output is not changed by a shifted input; shifting the input merely shifts the output by the same amount. For example, assume that a signal has time as the independent variable. A shift in this signal means a delay or advance by that shifting amount. A shift-invariant system, using this input signal delayed by an amount T, produces a delayed output:

$$f_1(t - T) \rightarrow g_1(t - T) \tag{2.6}$$

This shifting property applies to any other space where the signals are specified. For example, if an input 2D image is shifted (translated) relative to its origin, a shift-invariant system produces the same output image shifted by the same amount.

Most of the systems considered in signal processing have this shift-invariant property, but some interesting systems do not exhibit this behavior. A system is said to be *shift-variant* if a shifted input signal produces any signal other than just the shifted output signal. In Chapter 4, the interaction properties of light with matter will be discussed in terms of system properties, and certain materials will be shown to be shift-variant.

2.2 The Impulse Response of a System

The impulse response of a system characterizes how the system responds to an impulse signal: given an impulse signal (Equation (2.1)) as the input to a system, the *impulse response* is the output of that system. The impulse response of a system completely specifies a linear system. This impulse response property becomes evident if one realizes that any signal $f(x)$ can be represented as an infinite sum of shifted and scaled impulses. Then, if each of these shifted and scaled impulses is applied to the system, an impulse response is produced for each. The final superposition (or sum) of all these impulse responses represents the response of the system to the arbitrary signal $f(x)$. Mathematically, this property is represented with the sifting integral presented in Equation (2.3)—the response of a system to an impulse signal.

When dealing with optical and imaging systems, the impulse response is called a *point-spread function* (PSF) due to the type of impulse signal used with those systems. The impulse signal is the light isotropically emitted by a point source. For example, a simple lens (a system) taking as input a point source at the object plane produces a spot in the image plane (see Figure 2.2). This spot is the point-spread function of the system. An ideal lens produces an infinitely sharp spot when the image plane is in focus. When the image plane is not in focus, the spot takes the shape of a disk or an ellipse with radially-varying intensity, depending upon the orientation of the image plane. A non-ideal lens produces a warped version of the ideal spot, depending upon the aberrations influencing the impulse response of the lens/system. Arbitrarily shaped light sources emit more complicated wave fronts of

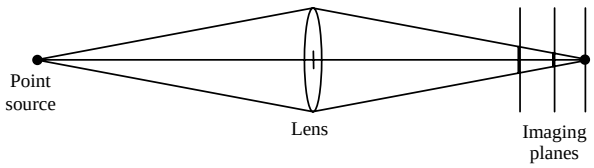


Figure 2.2: *A lens creating the image of a point source.*

light than the ideal point source. However, according to the Huygens-Fresnel principle [Ditchburn91], the propagation of any wave front can be decomposed into the propagation of waves produced by an infinity of point sources distributed along the original wave front. The envelope of all these point sources describes the position of the original wave front at any point in time along its propagation (Figure 2.3). In terms of signal processing, the response of an optical or imaging system to an arbitrary signal is the superposition of the corresponding impulse responses or point-spread functions,

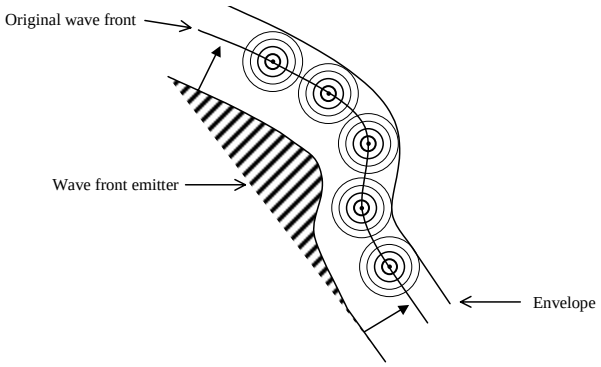


Figure 2.3: *Huygens-Fresnel principle applied to an irregularly-shaped wavefront emitter.*

as described by the sifting property Equation (2.3). Figure 2.4 illustrates the result of applying an arbitrary wave front (signal) discretized in point sources (impulse signals) to a lens (system). The envelope or superposition of all the resulting impulse responses approximates the response of the system to the original arbitrary input signal.

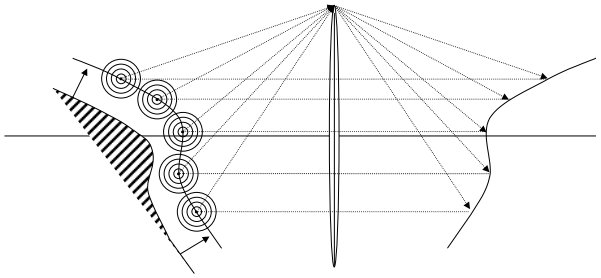


Figure 2.4: *Application of a decomposed arbitrary input wave front (signal) to a lens (system), and reconstruction of the same wave front on the image side.*

Thus, the response of a system can be represented as the superposition (or sum) of impulse responses. This property leads us to the concept of convolution.

2.3 The Convolution Operation

The *convolution operation*, $\star$, describes the output, $g(x)$, of a system, due to a given input, $f(x)$, applied to the system which has impulse response $k(x)$. Mathematically, convolution is expressed with the convolution integral, which in one dimension is

$$g(x) = f(x) \star k(x) = \int_{-\infty}^{+\infty} f(u)\, k(x-u)\, du. \tag{2.7}$$

For each output point, x, the convolution can be seen as the integration (gathering of contributions) of the input signal, $f(u)$, on an infinite neighborhood around the point x, weighted by the system's impulse response, $k(x-u)$, in that same neighborhood. Although Equation (2.7) integrates over an infinite space, in practice, the neighborhood is defined by the *support* or *width* of the convolution kernel $k(x)$—the region where the kernel has nonzero values. The kernel k can be seen as a sliding window that is shifted along the domain of the input signal. For evaluating the convolution at a point x, the kernel is centered at that location in the domain and the sum of the point-wise products of the two functions (input signal and kernel) is taken.

Notice the similarity between the convolution Equation (2.7) and the sifting Equation (2.3). Both equations integrate the product of two functions: the signal $f(x)$ and another function. This other function is the convolution kernel and represents the impulse response of the system. In the sifting equation the kernel reduces to a delta function and represents the impulse response of the identity system ($k(x) = \delta(x)$)—the system whose output always equals its input. In the convolution equation, the kernel represents the impulse response of an arbitrary system, as presented in Section 2.2.

In two dimensions, convolution is defined as

$$g(x,y) = f(x,y) \star k(x,y) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(u,v)\, k(x-u, y-v)\, du\, dv. \tag{2.8}$$

Several aspects of convolution will be analyzed in the sections that follow. In particular, Section 2.6.2 will describe graphically the convolution operation applied to the reconstruction of discrete signals, which will be helpful in understanding the convolution concept.

2.3.1 Circular and Hemispherical Convolution

Signals of the same dimensionality can be specified in a variety of spatial domains, depending on the units of the signals' independent variables. For example, in one dimension, Euclidean distance along a curve, chronological time, and angle around a circle have completely different shapes and interpretations but share the same dimensionality (Figure 2.5(a)). In two dimensions, rectangular coordinates, polar coordinates, and spherical coordinates also have different shapes and interpretations but share the same dimensionality (Figure 2.5(b)). The key element for signal processing is not the meaning or shapes of the domain, but the dimensionality of the signal. Signals of the same dimensionality are treated the same in signal processing theory.

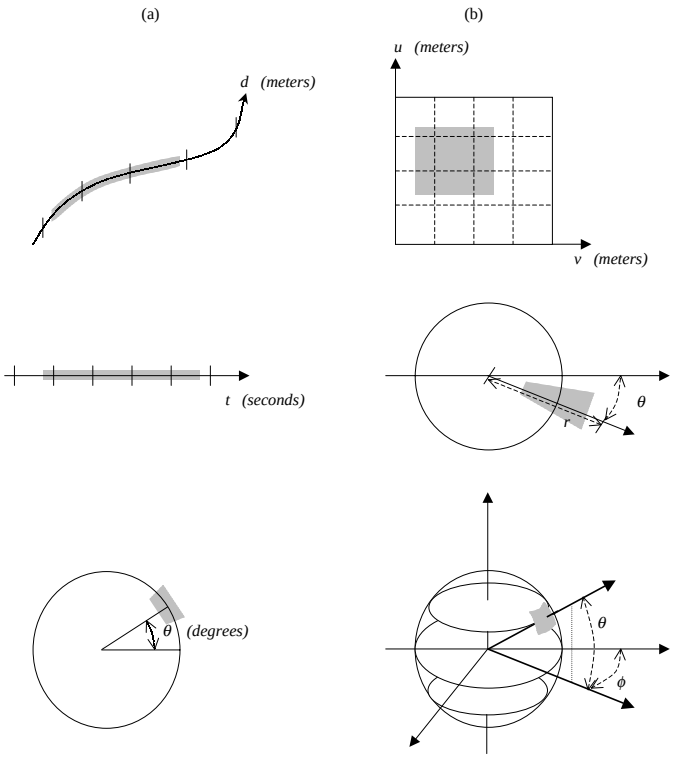


Figure 2.5: *Functions of same dimensionality: (a) one-dimensional; (b) two-dimensional.*

Of relevance to this dissertation, is convolution in circular, spherical, and hemispherical domains. Starting with a circular domain, the signal is a one-dimensional function of an angle, $f(\theta)$, and the convolution in that domain is given by

$$g(\theta) = f(\theta) \star k(\theta) = \int_{-\pi}^{+\pi} f(\alpha)\, k(\theta - \alpha)\, d\alpha. \tag{2.9}$$

Note the differences between this equation and Equation (2.7). Besides the change in independent variable, the integration is now performed in a circular domain of $[-\pi, +\pi]$, instead of in an infinite domain. By increasing the dimensionality of the domain and going to a spherical space, the same differences apply between the two-dimensional convolution in a rectangular domain—Equation (2.8)—and the convolution in the spherical domain: the independent variables x and y are replaced by two angles θ and ϕ, and the domain of integration becomes $\theta \in [-\pi, +\pi]$ and $\phi \in [-\pi/2, +\pi/2]$, instead of the infinite domain. Convolution in the spherical domain is then given by

$$g(\theta, \phi) = f(\theta, \phi) \star k(\theta, \phi) = \int_{-\pi}^{+\pi} \int_{-\pi/2}^{+\pi/2} f(\alpha, \beta)\, k(\theta - \alpha, \phi - \beta)\, d\alpha\, d\beta. \tag{2.10}$$

Of special interest in this dissertation is the convolution in hemispherical space. The only change implied in Equation (2.10), in order to convert it to hemispherical domain, is in the limits of integration

for the independent variable θ: it ranges in $[0, \pi]$ instead of ranging in $[-\pi, +\pi]$. Convolution in hemispherical domain is given by

$$g(\theta, \phi) = f(\theta, \phi) \star k(\theta, \phi) = \int_{-\pi}^{+\pi} \int_{0}^{+\pi/2} f(\alpha, \beta)\, k(\theta - \alpha, \phi - \beta)\, d\alpha\, d\beta. \tag{2.11}$$

The shaded regions in Figure 2.5 illustrate placement of convolution kernels in the specific domains (recall the window analogy of convolution kernels from Section 2.3).

2.3.2 Spatially Variant Convolution

So far, the presentation of convolution has assumed spatially invariant kernels, i.e., that the base shape and the weighting function of the kernel does not vary with respect to the independent variable. In a more general approach, convolution can be presented with a spatially variant kernel. In such a case, the kernel shape varies across the domain of integration. In one dimension, this dependence of the kernel is represented with a function $w(x)$. The convolution Equation (2.7) then becomes

$$g(x) = f(x) \star k(x, w(x)) = \int_{-\infty}^{+\infty} f(u)\, k(x - u, w(x))\, du \tag{2.12}$$

and in two dimensions

$$g(x, y) = f(x, y) \star k(x, y, w(x, y)) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(u, v)\, k(x - u, y - v, w(x, y))\, du\, dv. \tag{2.13}$$

Spatially variant convolution comes up for several systems. In practice, most physically realizable systems are spatially variant. For example, most lenses and optical imaging systems have finite image area due to aberrations—images of on-axis objects are correct, but images of off-axis objects are deformed. In Chapter 4, surface material properties will be discussed as a system in terms of interaction with light, and it will be seen that most materials tend to exhibit spatially variant behavior.

Section 2.6.2.3 presents graphically the application of spatially variant convolution to reconstruction, which may be helpful in understanding this type of convolution.

2.3.3 Discrete Convolution

Equation (2.7) is defined for continuous functions $f(x)$ and $k(x)$. However, in computer science the input signal and the convolution kernel are, in general, discrete functions. This demands a discrete convolution expressed with the following summation:

$$g(x) = f(x) \star k(x) = \sum_{i=-\infty}^{+\infty} f(i)\, k(x - i) \tag{2.14}$$

where x and i take only integer values. In two dimensions, discrete convolution is given by

$$g(x,y) = f(x,y) \star k(x,y) = \sum_{i=-\infty}^{+\infty} \sum_{j=-\infty}^{+\infty} f(i,j)\, k(x-i, y-j). \tag{2.15}$$

2.4 The Frequency Domain

All previous discussion has taken place in the spatial domain; the independent variables of our signals were always time, Euclidean coordinates, polar coordinates, etc. So far, an arbitrary signal in a spatial domain has been considered to be the sum of an infinity of scaled and shifted impulses. Alternatively, Fourier proposed that a signal could also be represented as an infinite sum of sinusoidal waves of a base frequency and all its multiples. Both representations capture the same information but encode it in different forms. The advantage of using the dual domains to represent signals or systems is that sometimes it is easier to understand and manipulate a signal or a system in one domain than it is in the other. In the next section, the Fourier transform—the tool that converts from a spatial domain to the frequency domain and vice-versa—will be discussed. Then, Section 2.4.2 discusses the convolution theorem, which relates two operations—convolution and multiplication—in the two dual domains—spatial and frequency domains.

2.4.1 The Fourier Transform

The *Fourier transform* is the tool which allows conversion between spatial and frequency domains. The Fourier transform of a one-dimensional, continuous function $f(x)$ is defined as

$$\mathcal{F}\{f(x)\} = F(\nu) = \int_{-\infty}^{+\infty} f(x)\, e^{-j2\pi\nu x} dx \tag{2.16}$$

where $j^2 = -1$, the frequency variable ν is measured in cycles per unit of x, and $e^{\pm j2\pi\nu x} = \cos(2\pi\nu x) \pm j \sin(2\pi\nu x)$. The *inverse Fourier transform* of $F(\nu)$ is defined as

$$\mathcal{F}^{-1}\{F(\nu)\} = f(x) = \int_{-\infty}^{+\infty} F(\nu)\, e^{j2\pi\nu x} d\nu. \tag{2.17}$$

Note that the only difference between the direct and inverse transformations is the sign of the exponent. The functions $f(x)$ and $F(\nu)$ are called a *Fourier transform pair*; and for any function $f(x)$ the Fourier transform is unique, and vice-versa. The Fourier transform of a signal is called the *spectrum* of that signal, and the inverse Fourier transform of a spectrum generates its corresponding signal. The Fourier transform of the impulse response of a system is called the *transfer function* of that system—the system's *frequency response*.

The Fourier transform (Equation (2.16)) decomposes a signal $f(x)$ into an infinite sum of complex exponentials (sinusoidal waves) of a base frequency and its multiples. The complex function $F(\nu)$ represents the amplitude and phase of each complex exponential, for each frequency value ν. The Fourier transform has several other properties and the interested reader is referred to [Oppenheim96, Castleman96] for a complete presentation of this topic.

If we discretize both the spatial and the frequency domains, the discrete Fourier transform becomes

$$F_n = \frac{1}{N} \sum_{i=0}^{N-1} f_i \; e^{-j2\pi \frac{n}{N} i} \tag{2.18}$$

and the inverse discrete Fourier transform is

$$f_i = \frac{1}{N} \sum_{n=0}^{N-1} F_n \; e^{j2\pi \frac{i}{N} n}. \tag{2.19}$$

where N is the length of the discrete sequence f_i, and $0 \leq i, n \leq N-1$ are indices.

2.4.2 The Convolution Theorem

The *convolution theorem* states that convolution in one domain is equivalent to multiplication in the other domain, i.e.: convolution in the spatial domain is equivalent to multiplication in the frequency domain and multiplication in the spatial domain equals convolution in the frequency domain. As will be shown in the next sections, the convolution theorem has important applications in filtering, and in sampling and reconstruction. The convolution theorem also influences how signal processing should be computed for discrete signals. Based on the computational cost of convolution and multiplication, and depending on the size of the data set, it may be less expensive to perform either convolution or multiplication in the corresponding dual domains.

2.5 Filtering

Several applications demand changing the frequency content of a signal. Modifying the relative amplitudes of the frequency components and eliminating some frequency components of a signal are examples of such processing. This operation is called *filtering*, and a *filter* is a system that takes an input signal $f(x)$ and produces an output signal $g(x)$. This definition resembles the convolution operation from Section 2.3 with the convolution kernel representing the filter in the spatial domain. In fact, filtering in the spatial domain is performed through convolution. Alternatively, in the frequency

domain, filtering is carried out through multiplication, according to the convolution theorem of Section 2.4.2. Given this duality, filters can be designed either in the spatial domain or in the frequency domain.

Analysis in the frequency domain is more intuitive; a filter in the frequency domain is a function that represents the amplitude modulation of each frequency value desired in the filtered output signal. The identity filter has a constant value of unity along the entire spectrum and multiplication of this filter with a signal in the frequency domain does not change the signal. A filter that halves the amplitude for a given frequency value has a value of one-half at that particular frequency, and so on. Then, the filtering operation in the frequency domain is performed by a point-wise multiplication of the filter value by the corresponding signal value. Alternatively, the Fourier transform of a frequency domain filter can be used to perform the filtering through convolution in spatial domain. The convolution of the signal and the filter in the spatial domain produces the same results as multiplication in the frequency domain.

2.6 Sampling and Reconstruction

Rendering a computer generated image involves the application of multiple resampling processes. A *resampling process* starts with discrete data, reconstructs a continuous representation of the same information, and then samples this continuous representation to generate the resampled version of the data (Figure 2.6). This resampling process is repeated at different scales and stages of the graphics pipeline. For example, the polygon rendering process is a resampling process: it starts with a set of discrete samples (vertices); reconstructs a continuous representation (polygons); and samples the polygons to create a new, resampled representation (pixels). Similarly, curved-surface rendering and image-based rendering involve the same resampling paradigm. Shading is also a resampling process: attributes (color, texture coordinates, etc) are known at vertices of a geometrical model and form the set of discrete samples; reconstruction takes place by performing interpolation (nearest neighbor, linear, or higher order) between samples; and the final sampling occurs by outputting the interpolated values at the pixels. Texture mapping operates in the same interpolated way, but in texture space, instead of image space.

Sampling and reconstruction are related by a theorem; the *sampling theorem* states that, if a continuous signal is sampled finely enough, it can be recovered completely using reconstruction techniques. In more detail, given the frequency representation of a signal, if the sampling rate is at

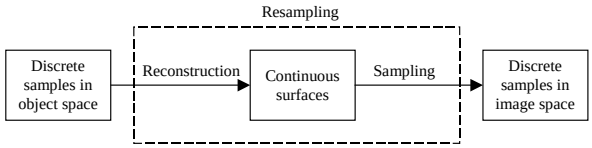


Figure 2.6: *Resampling process.*

least twice as high as the highest frequency in the signal, the original signal can be reconstructed by using ideal reconstruction filters.

2.6.1 Sampling

Sampling is the process that converts a continuous signal, $f(x)$, into a set of discrete samples, $f_s(x)$. Figure 2.7 illustrates the sampling of a one-dimensional function in the spatial domain. Mathematically, sampling in the spatial domain is represented by the multiplication of the continuous

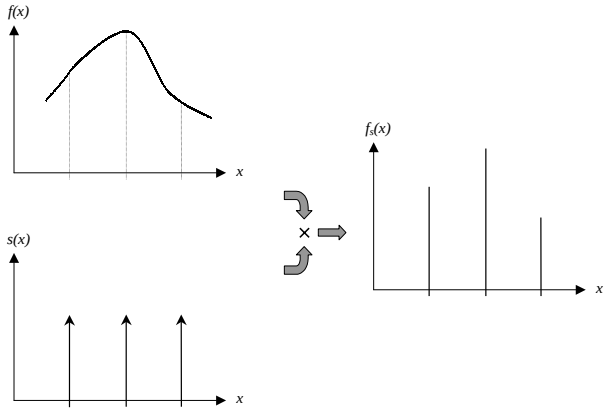


Figure 2.7: *Sampling in the spatial domain.*

signal by the comb function, $s(x)$:

$$f_s(x) = f(x) \times s(x) \tag{2.20}$$

where the comb function is a train of evenly spaced delta functions:

$$s(x) = \sum_{n=-\infty}^{+\infty} \delta(x - n). \tag{2.21}$$

Through the convolution theorem, the sampling process can also be specified in the frequency domain. The continuous input signal and comb function can be each represented in the frequency domain with $F(\nu)$ and $S(\nu)$, respectively. This new representation is given by the Fourier transform

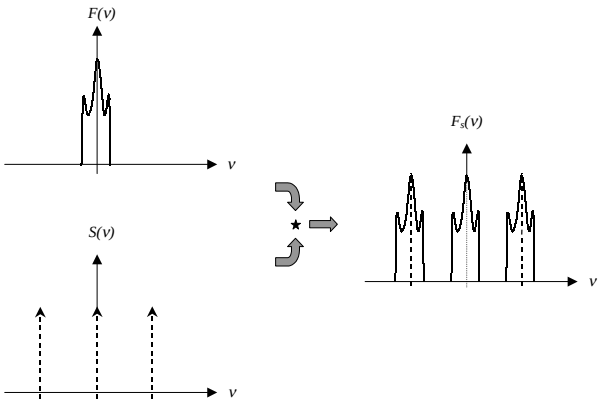


Figure 2.8: *Sampling in the frequency domain.*

of the functions: the Fourier transform of the comb function looks like another comb function. The separation of pulses of both representations is related by Fourier transform properties; if the comb function has a period of τ in spatial domain, its pulse spacing in frequency domain is $\frac{1}{\tau}$. Sampling in the frequency domain, illustrated in Figure 2.8, is given by the convolution of the frequency response of the system, $F(\nu)$, and the Fourier transform of the comb function, $S(\nu)$:

$$F_s(\nu) = F(\nu) \star S(\nu) \tag{2.22}$$

Note that a sampled signal in the spatial domain is a pulse-based representation of the continuous signal, whereas a sampled signal in the frequency domain represents the replication of its continuous frequency response. The frequency spectrum of the sampled signal ($F_s(\nu)$ in Figure 2.8) looks like the frequency spectrum of the continuous signal ($F(\nu)$ in Figure 2.8), but it is replicated at multiples of the sampling frequency (the spacing of the comb function in frequency domain—$S(\nu)$ in Figure 2.8). Although it is more intuitive to understand and visualize sampling in the spatial domain, its representation in the frequency domain is more useful in terms of signal analysis, for example, in anti-aliasing applications.

2.6.2 Reconstruction

Reconstruction is the process that generates a continuous signal from a set of discrete samples. Reconstruction can be considered to be a mapping operation: given a set of discrete values in one domain (input domain), reconstruction finds a continuous function in another domain (output domain). This mapping can be performed either with a feed-backward approach—gathering information from the input domain for each point in the output domain—or with a feed-forward approach—spreading

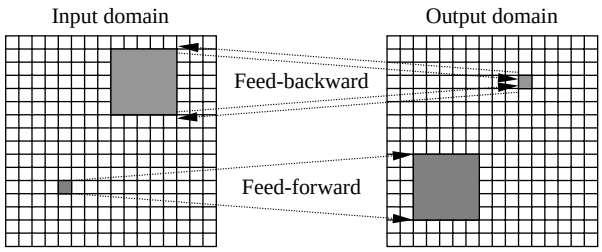


Figure 2.9: *Mapping: feed-backward and feed-forward approaches. Feed-forward mapping takes information directly from the input domain into the output domain; whereas feed-backward mapping starts the process in the output domain, queries the input domain, and finally takes the information from the input domain into the output domain.*

the contribution of each point from the input point into the output domain (see Figure 2.9). Both approaches reach the same results; but, in general, one is easier to understand and involves less computational cost than the other. Westover [Westover91] compares feed-backward and feed-forward convolution methods in terms of accesses to the input and output signals, and gives an example illustrating the fewer number of combined accesses required by feed-forward convolution, compared to feed-backward methods.

2.6.2.1 Feed-backward reconstruction

In a *feed-backward* approach, for each desired value in the output domain, information is gathered from the input domain. Both interpolation and straight convolution are instances of feed-backward reconstruction. In fact, convolution is the more general approach, and interpolation can be expressed in terms of convolution with the appropriate choice of kernels [Wolberg92].

Interpolation

Interpolation is a polynomial-based reconstruction scheme. Given a sampled signal, interpolation fits a continuous curve to the data and reconstructs parts of the signal lost in the sampling process. At each desired new point in the domain, interpolation evaluates the interpolant polynomial. For example, in one dimension, linear interpolation connects every two adjacent points in the data using a linear or first-order polynomial. Figure 2.10(a) shows the discrete data and Figure 2.10(b) presents the linearly interpolated reconstruction.

Higher-order interpolants require using more information in a local neighborhood than just the two adjacent points. For example, Figure 2.11 presents second-order or quadratic interpolation of the same data used in Figure 2.10: a point in the interpolated function is the result of the weighted sum of the three closest data samples.

(a) (b)

Figure 2.10: *Linear interpolation of sampled data.*

(a) (b)

Figure 2.11: *Quadratic interpolation of sampled data.*

The higher the interpolation-order, the wider the local neighborhood, and the more control there is about the derivatives of the interpolated function along its domain. Note that a linearly reconstructed function, also called *piecewise linear*, is continuous in value along its domain, but it is discontinuous in first derivative at the sample points. This type of function is usually called a C^0-continuous function. Note also that piecewise quadratic functions are not enough to provide C^1-continuous functions—continuous up to first derivative along their entire domain. Although independent pieces of a piecewise quadratic function can be C^1-continuous, when three or more quadratic pieces are put together it may not be possible to satisfy the first derivative continuity criteria at all the junction points[1]. A piecewise cubic function is necessary to provide full C^1 continuity along the entire domain. Higher-order continuity is possible, but usually not necessary in computer graphics applications.

Convolution

Convolution was presented mathematically in Section 2.3. Now, convolution will be examined graphically through its application to the reconstruction of a discrete signal. Both a feed-backward and

[1]The following algorithm for finding the two quadratic pieces for three samples of a one-dimensional function cannot ensure C^1 continuity at the middle point:

- select a quadratic that satisfies the value of the samples at the left and at the middle point, and the slope at the left point
- select a quadratic that satisfies the value of the samples at the middle and at the right point, and the slope at the right point

This problem generalizes for any number of points greater than two and when the slope selection algorithm does not follow a sequential order of the data. Note that the algorithm above would always find a C^1-continuous quadratic interpolation if the slopes at the sampling points were always selected from left to right or vice-versa.

a feed-forward approach to convolution of a discrete signal with a triangular kernel will be discussed. They reach exactly the same result as linear interpolation, as presented above.

In feed-backward convolution, for each point in the output domain, information is gathered (integrated) from the input signal. The final output value at each point is the weighted average of the input values covered by the kernel neighborhood centered at the input point. In Figure 2.12,

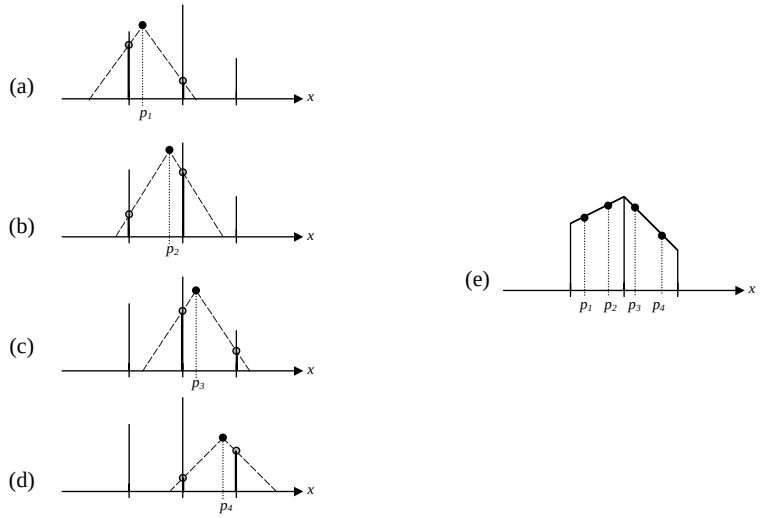


Figure 2.12: *Feed-backward convolution.*

the same sampled signal from the linear interpolation example (Figure 2.10(a)) is convolved with a triangular kernel. Figures 2.12(a)-(d) represent the evaluation of the output value for particular points in the output. Imagine for each point p in the output, the instantiation of the triangle kernel centered at that same location in the input signal (points p_1 to p_4 in the figure). Now, for all the points in the input covered by the kernel, add the product of the corresponding kernel value and input signal value. For example, the instantiation of the triangular kernel at point p_1 in the input domain (Figure 2.12(a)) overlaps the two left-most samples of the sampled signal. The intersection point of the kernel with the sample spikes corresponds to the contribution of each of those samples to the final result at point p_1. The resulting value (convolution) at point p_1 in the output domain (Figure 2.12(e)) is the sum of the heights of the intersected samples from the input domain. This results in the convolved output signal (e). The pseudo-code for feed-backward convolution is:

```
for each point p in the output
        center kernel in the input at the same location p
        set output value at p to 0
        for each point q in the input covered by the kernel
                output value at p += input value q * kernel value at q
```

2.6.2.2 Feed-forward convolution — slatting

An alternatively to the feed-backward approach as presented above, is the feed-forward approach. Each value in the input domain spreads its information in the output domain. In Figure 2.13, a sampled

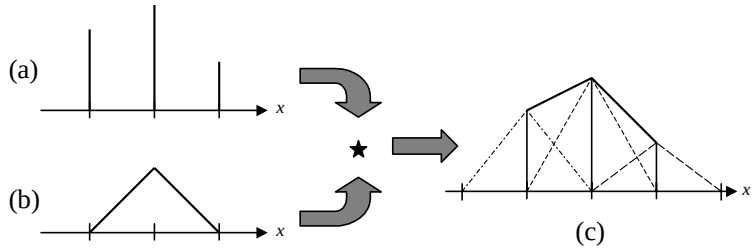


Figure 2.13: *Feed-forward convolution (splatting): (a) discrete input signal, $f(t)$; (b) triangular kernel, $k(t)$; (c) convolved output signal, $g(t)$.*

signal (a) is convolved with a triangular kernel (b). The result is the convolved signal in (c). Think of (c) as the instantiation of the triangle kernel (b) centered at each sample of the discrete signal (a). The amplitude of each sample in the input signal (a) gives the height of the corresponding instantiated triangle in the output signal (c). The support or width of the kernel is the same (invariant) for all the samples. The superposition (or sum) of the contributions of instantiated kernels results in the convolved output signal (c). The pseudo-code for feed-forward convolution is:

```
zero all output points
for each point q in the input
        center kernel in the output at the same location q
        for each point p in the output covered by the kernel
                output value at p += input value at q * kernel value at p
```

The reconstruction result obtained with linear interpolation in Figure 2.10 is exactly equal to the reconstruction result obtained with convolution using a triangular kernel both with feed-backward and with feed-forward approaches. Interpolation with higher-order polynomials also has corresponding kernels to produce equivalent results using convolution. Note, however, the difference in computational cost obtainable with the feed-forward pseudo-code in comparison with the feed-backward one. The feed-forward approach has a reduced number of combined accesses to the input and output in the inner loop. This indicates that, in some situations, the feed-forward algorithm may be a faster approach than the feed-backward convolution. When the number of points in the input and output domains is the same (for example, in a filtering operation), the forward approach is clearly more efficient in terms of

combined accesses. However, when the number of desired points in the output is much smaller than the number of points in the input, the backward approach is more efficient. A detailed analytical analysis of these tradeoffs is possible, but not fundamental to this dissertation.

2.6.2.3 Reconstruction with spatially variant convolution

As Section 2.3.2 mentions, sometimes the convolution kernel is allowed to change in terms of shape and weighting function along the domain of convolution/integration. The graphical interpretation of discrete spatially variant convolution is presented in Figure 2.14. A sampled signal (a) is convolved with a spatially varying triangular kernel (b), as in the previous sections. However, in this section let the kernel width vary with the discrete function presented in (d), which represents a scaling factor for the kernel width along the parameter x. In this example, it represents a linearly increasing scaling factor along the parameter x; the spatial variance of the kernel is only in terms of its width. The result is the convolved signal in (c), which represents the summation of the instantiations of triangular kernels with width given by the scaling function (d). Notice that the leftmost instantiation of the kernel is the narrowest, while the rightmost is the widest, in accordance with (d). The function in (d) controls the region of influence/spread of each point from the input signal (a).

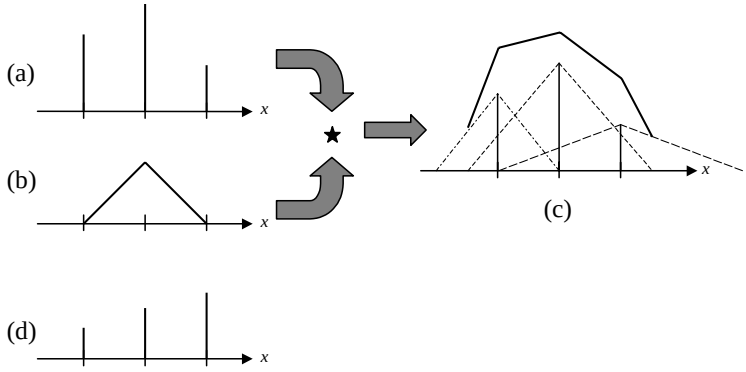


Figure 2.14: *Spatially variant convolution: (a) discrete input signal, $f(x)$; (b) triangular kernel, $k(x, w(x))$; (c) convolved output signal, $g(x)$; and (d) the function representing the kernel width along x.*

This varying region of influence of the kernels is key in reconstructing depth- and direction-dependent functions, such as for integrating the radiance equation, as will be seen in Chapter 4. Note, however, that the varying regions of influence of the kernels may introduce discontinuities in the reconstructed signal. Notice, for example, the first derivative discontinuities in the reconstructed signal introduced at the extreme points of the kernel instantiated at the central sample in Figure 2.14.

CHAPTER 3

THE LIGHT TRANSPORT PROBLEM

Light is a form of energy that can be transferred from one place to another. It is the form of energy that enables us to see—or to which our eyes are sensitive—and that all rendering techniques try to simulate in computer graphics. This chapter reviews important concepts related to light, matter, and their interactions in the real world. Most of these concepts derive from *radiometry*—the physical science that studies the measurements of electromagnetic radiation. The chapter starts with a qualitative description of light transport and culminates with the light transport equation, which determines the light leaving a point in a particular direction in a scene due to all possible light transfers in the environment. The interested reader is referred to [Ditchburn91, Feynman89] for detailed expositions of this field and to [Watt92, Cohen93, Sillion94, Glassner95] for its application in image synthesis.

3.1 Light

Light is the visible part of electromagnetic radiation—sinusoidal waves created by coupled magnetic and electric fields. The *electromagnetic spectrum*, defined in terms of wavelength, ranges from meter-sized radio waves down to nanometer[1] scale x-rays. The wavelength of visible light is perceived as color by our eyes, and it varies from approximately $400nm$ for violet to about $700nm$ for red. *Monochromatic* radiation is composed of waves of a single wavelength, whereas more complex light is composed by the superposition of several wavelengths. Electromagnetic radiation can also be *coherent*: the wavefront of light, as it propagates, stays in phase at all points in space and time. As with any other wave, light waves also interact among themselves and form interference patterns. Additionally, electromagnetic radiation can be polarized—show a preferential orientation of the field vectors with respect to a fixed vector perpendicular to the direction of light propagation. Unpolarized radiation

[1] $1\ nanometer = 1nm = 1 \times 10^{-9}m.$

involves the superposition of waves with random orientations, whereas polarized light consists of waves with a preferred orientation, e.g., linear and circular polarization [Fowles89].

Although we presented light in terms of waves, it can also be understood in terms of rays and in terms of particles called *photons*. Each of these three different representations of light applies in different situations. There are mechanisms to explain and unify these natures [Feynman89], but this discussion will focus on ray optics, due to its greatest influence in image synthesis. To justify emphasizing geometrical optics, the three light propagation mechanisms and their limitations will be analyzed.

3.2 Optics

Optics is the branch of the physical sciences that studies light. Optics is typically divided into three subfields—geometrical optics, physical optics, and quantum optics—depending on the relative wavelength of the light, or its energy, with respect to the dimensions of the light measuring equipment, or its sensitivity [Feynman89]:

- *Geometrical optics*, also called *ray optics*, represents light propagation with rays and applies whenever the wavelengths and the photon energies are negligibly small compared to the size and sensitivity of the light measuring equipment. Ray optics is important (when the size of objects is larger than the wavelength of light) for understanding macroscopic properties of light such as shadows, reflection, and refraction.

- *Physical optics*, also called *wave optics*, represents radiation with waves and applies whenever the wavelengths are comparable to the dimensions of the measuring equipment but the energy of the particles is still small compared to the sensitivity of the equipment. Wave optics is necessary (when the size of the objects is comparable to the wavelength of light) for understanding properties such as polarization, diffraction, interference, and holography.

- *Quantum optics*, also called *particle optics*, represents light with particles—*photons*—and applies whenever the wavelengths are negligibly small compared to the dimensions of the measuring equipment and the particle energies are much greater than the equipment sensitivity. Particle optics is needed (when the size of the objects is smaller than the wavelength of light) to understand the microscopic properties of light such as the interaction of light with atoms

and molecules. The theory of particle optics is necessary for understanding phenomena such as fluorescence, phosphorescence, dispersion, and light amplification (e.g. LASER).

Visible light can be described with any of the approaches above, but, unless microscopic effects are needed, geometrical optics is enough to explain most of the effects of light. In terms of image synthesis, geometrical optics is usually sufficient to simulate macroscopic effects. However, effects such as interference, fluorescence, and phosphorescence have already been simulated in computer graphics [Glassner94, Gondek94] without appealing to complete wave and quantum theories. By restricting the analysis to geometrical optics, i.e., if the wave nature and the particle nature of light are not considered, the following microscopic phenomena are not accounted for:

- Polarization, and hence some amplitude attenuation due to polarized reflections [Fowles89]
- Interference, and hence thin film color effects/phenomena such as in soap bubbles [Gondek94]
- Diffraction, and hence holography [Collier71]
- Dispersion, and hence rainbows [Greenler80].

Note that all these microscopic phenomena have second-order impact compared to the first-order impact of the macroscopic effects simulated with geometrical optics.

3.3 Reducing the Dimensionality of the Problem

From the discussion so far, light transport is a multi-dimensional problem. Section 3.2 eliminated some of the variables of the problem by restricting the domain to geometrical optics. However, the remaining problem is still dependent on several dimensions such as position, orientation, time, wavelength of light, and the refractive index of participating media. Although the solution of the full problem would produce ideally realistic results, simplifying assumptions are necessary to reduce the dimensionality of the problem and to reduce the computational complexity of the solution methods. This section eliminates some of the remaining variables, which are not fundamental from the image synthesis perspective.

On the scale of human perception, light interacts with matter instantaneously and what our eyes perceive is an energy equilibrium inside a scene. The discussion of the light transport equation in this dissertation ignores all dependence on time—it approximates a steady state solution.

As seen in Section 3.1, light is defined in a continuous domain. It is the visible part of the electromagnetic radiation. This means that wavelength is one of the variables of the light transport problem. Instead of fully considering wavelength in the light transport equation, image synthesis, in general, only samples this variable for three particular distributions of wavelengths—*red, green, and blue*—which match the color sensitivities of the human eye receptors [Foley90]. We assume that the wavelength of light is not changed by interaction with matter, as in fluorescence. The distributions of wavelengths do change, as when light is partially reflected or absorbed by a colored medium.

Although ignored in our discussion, light transport is usually affected by participating media, for example, air, water, glass, etc. A participating medium may produce absorption, scattering, and emission along a path of light. Despite its importance for effects like fog, clouds, and smoke we are going to assume that light transport happens in non-participating media such as clear air or vacuum.

3.4 Emission

Certain materials, under particular conditions, emit light. This section discusses the kinds of emission patterns to be expected from light sources, as opposed to how the light is created.

In terms of waves, a *point source* generates a spherical wave of light centered at the source. In terms of rays or particles, this means that rays or particles are radially emitted from the point source. More complex light source shapes generate wavefronts that can be described with the Huygens-Fresnel principle mentioned in Section 2.2: the complex wavefront can be decomposed into point sources, and the envelope of the point wavefronts reconstructs the original wavefront at any point in time along the wave propagation.

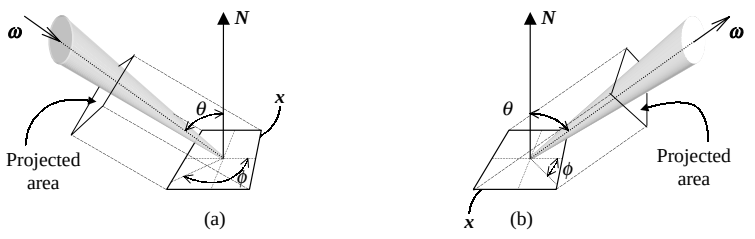


Figure 3.1: *Incoming and outgoing radiance at point* x *and direction* $\vec{\omega}$.

In summary, light sources can be represented as functions of direction around the source. In terms of radiometry the emission functions represent the radiance[2] leaving light source's point $\mathbf{x}$ in direction ω: $L_e(\mathbf{x}, \vec{\omega})$.

3.5 Light-Matter Interaction

The discussion so far has concentrated on light propagation between two points in a vacuum. However, light interacts with matter. To analyze light transfers in an environment, it is necessary to understand how light responds to materials. We need to understand materials' appearance. Instead of looking at the physical processes of light-matter interaction, this discussion will take a functional approach and describe a material or a medium in terms of how light responds to it, without looking at the internal process that generates that behavior.

When analyzing the interaction of light and matter, two cases arise. Light flowing through a medium can be absorbed or scattered by the propagation medium. And propagating light can change its behavior when it encounters the boundary between two media. At the boundary, light can be either reflected back into the original medium or transmitted into the new medium or both. By conservation of energy, if a certain amount of energy reaches a material, the sum of absorbed, reflected, and transmitted light must equal the incoming amount of light.

3.5.1 Ideal Reflection

Reflection is the phenomenon by which incident radiation on the boundary between two media returns to the initial medium. For an ideal mirror—a surface that, for a single incoming ray, generates a single reflected outgoing ray—the *laws of reflection* state that (see Figure 3.2):

- The reflected ray $\mathbf{R}$ lies on the plane defined by the incident ray $\mathbf{L}$ and the surface normal $\mathbf{N}$.
- The angle of the reflected ray with respect to the surface normal, θ_o, is equal to the angle of the incident ray with respect to the same normal vector, θ_i, at the same point of interest.

[2] *Radiance* is the power per unit projected area perpendicular to the direction of propagation per unit solid angle in the direction of light propagation. For a point $\mathbf{x}$ on a differential area dA, the radiance in direction $\vec{\omega}$, $L(\mathbf{x}, \vec{\omega})$, represents the amount of energy, per unit time, flowing at some point inside the solid angle with apex at $\mathbf{x}$ and direction $\vec{\omega}$, and passing through the projected differential area in the direction perpendicular to $\vec{\omega}$ (Figure 3.1). The flow inside the solid angle can represent outgoing or incoming energy. Outgoing radiance represents light emanating from point $\mathbf{x}$, or how the point illuminates the environment. Incoming radiance represents light reaching point $\mathbf{x}$, or what is visible from that point.

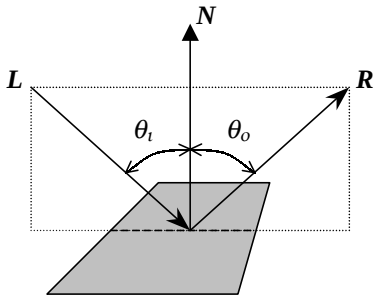


Figure 3.2: *Ideal reflection.*

For a general reflective material, the concepts of scattering are necessary, as will be seen in Section 3.5.3. In nature, energy along a single incoming ray can generate outgoing energy in any direction on the hemisphere above the reflective surface according to some directional distribution function.

3.5.2 Ideal Transmission

Transmission is the phenomenon by which incident radiation on a media boundary crosses the boundary into the new propagation medium. *Refraction* refers to the change in direction of travel as radiation passes from one medium to another. This change in direction is caused by the difference in the speed of light propagating through the two media. The *index of refraction* of a material represents the ratio of speed of light in the material to the speed of light in vacuum. For an ideally transparent

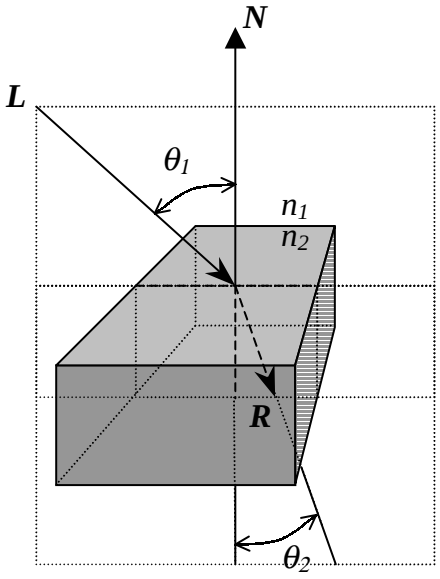


Figure 3.3: *Ideal transmission.*

material—a surface that, for a single incoming ray, generates a single transmitted outgoing ray—the *laws of transmission* state that (see Figure 3.3):

- The refracted ray $\mathbf{R}$ lies on the plane defined by the incident ray $\mathbf{L}$ and the surface normal $\mathbf{N}$.
- The relationship between the angle of incidence and the angle of refraction is given by *Snell's law*:

$$n_1 \cos\theta_1 = n_2 \cos\theta_2 \tag{3.1}$$

  where n_1 and n_2 are the indices of refraction of the two media, and θ_1 and θ_2 are the angles of the incident and refracted vectors with respect to the surface normal and its opposite vector, respectively.

- When light is passing from a more refractive medium to a less refractive medium, $n_1 > n_2$, there is a phenomenon called *total reflection*. Past a critical angle given by $\arcsin(\frac{n_2}{n_1})$, the incident light is completely reflected, instead of being refracted. This is the basic principle behind *fiber optics* [Fowles89].

Analogously to reflection, for a general transmissive material, the concepts of scattering are necessary, as will be seen in Section 3.5.3. In nature, energy along a single incoming ray can generate outgoing energy in any direction on the hemisphere below the refractive surface according to some directional distribution function.

3.5.3 Surface Roughness

So far, ideal reflective and ideal refractive materials have been discussed—when a single incoming ray interacting with such a material generates a single reflected or refracted ray. In nature, most materials are not ideal, and a scattering process happens at the boundary between two media. Light interacting with non-ideal materials can generate reflected and refracted rays in the entire reflection and transmission hemispheres, respectively. Also, for ideal materials it was implicitly assumed that all the energy would be reflected or transmitted and that the magnitude of energy being scattered or absorbed was not taken into account. In the real world, though, microscopic processes take place when light is reflected/transmitted by a material and only part of the incoming light is reflected back into the environment or transmitted into the new medium. This section will analyze the end results of scattering, as opposed to the physical processes involved.

Consider light reflection/refraction at a point on a surface. Assume that, microscopically, the surface is composed of randomly oriented microfacets (Figure 3.4). The average normal of the facets

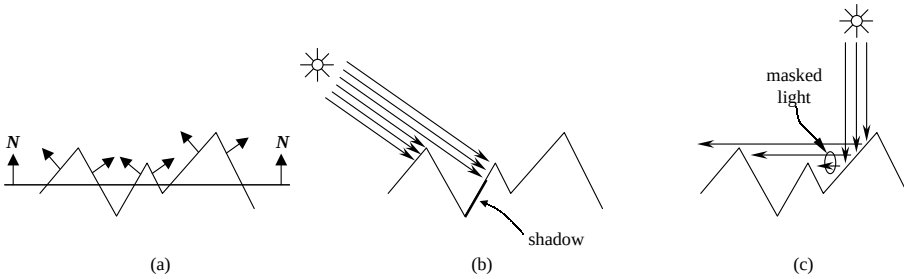


Figure 3.4: *Microfacets model: (a) displacement of facets' normals with respect to the average surface normal; (b) light shadowing; (c) light masking.*

represents the surface's normal vector $\mathbf{N}$ (Figure 3.4(a)). The size of the facets and the displacement of the facets' normals with respect to the average normal determine the roughness of the surface. A mirror has null displacement—is ideally polished—and increasing roughness implies higher amplitude and/or higher frequency displacements. Each of the microfacets, independently, reflects/transmits light according to the ideal model discussed above; it is an ideal mirror or an ideal refractor. However, at a larger scale, light reaching a facet can cast a micro-shadow (Figure 3.4(b)) and light leaving a facet can be masked (Figure 3.4(c)), due to the facets' boundaries. The random orientation of the reflective/transmissive facets can cause a single incident ray to be reflected/transmitted in any direction on the reflection/transmission hemisphere centered at the point of incidence other than the ideal one. The distribution of unidirectional light into a hemisphere and the shadowing and masking effects reduce the magnitude of the reflected/transmitted light with respect to the incident light.

As seen above, a single ray of incoming light can generate a new ray in any reflected/transmitted direction with the same or reduced energy. This non-ideal behavior creates an entire spectrum of materials other than the ideal reflectors and ideal refractors studied so far (Figure 3.5). On one end, there is the ideally *specular transfers* case that refers to energy concentrated on a single direction. On the other end, there is the ideally *diffuse transfers* case that corresponds to an equal distribution of energy in all directions. In between, there are the non-ideal—or *glossy*—transfers that refer to the continuum between perfectly specular and perfectly diffuse transfers. Glossy transfers have a directional bias but are neither restricted to a single direction nor impose an equal scattering out to a whole hemisphere. In this dissertation, *specular transfers* is used as a generalized term for non-diffuse transfers, including both glossy and ideally specular transfers. Although the terms diffuse and specular

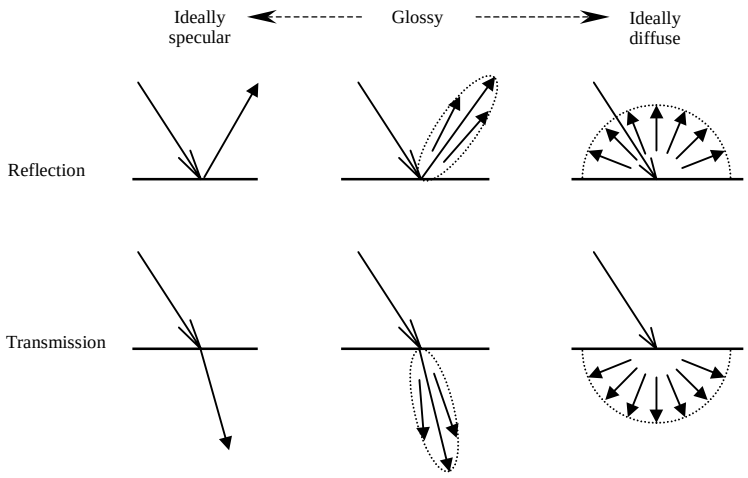


Figure 3.5: *Light-matter interaction for reflection and transmission, ranging from ideally specular transfers to ideally diffuse transfers.*

are implicitly related to reflection, transmission will be classified with similar terminology in this dissertation. For example, the transmission by an ideal refractor will be called a *specular refraction*.

The two extremes of this spectrum represent important material properties: the ideally diffuse end represents light transfers that do not vary their magnitudes with orientation, whereas all the other transfers in the spectrum represent increasing variance of magnitude for increasing specularity. This property will prove useful for computing light transport for image synthesis.

3.5.4 Anisotropy

Recall, from the definitions of reflection and refraction for ideal materials in Section 3.5, that the outgoing rays were both dependent on a single angle—the angle of the incoming ray with respect to the surface normal. However, there is another degree of freedom that needs to be taken into account: the rotation around the surface normal (see the angle ϕ in Figure 3.1).

All previous definitions assumed *isotropic* materials—reflection and refraction were invariant with respect to rotation of the surface around the surface normal vector. However, some materials in nature present *anisotropic* behavior. For those materials, reflection and refraction properties vary with respect to rotations of the surface around its normal vector. For example, brushed metals, cloth, and hair exhibit anisotropic reflections.

3.5.5 Bidirectional Distribution Functions

The discussion so far has concentrated on qualitative analyses. This section discusses a quantitative approach to reflection and gives hints for extending the approach to transmission. As is usual in physics and graphics, transmission is left as a straightforward extension of reflection, given the high similarity of the two phenomena.

Consider light reflection at a point $\mathbf{x}$ on a surface. The amount of outgoing or reflected light in a direction $\vec{\omega_o}$ is directly proportional to the incident light from a solid angle centered in direction $\vec{\omega_i}$. Increasing the solid angle or increasing the density of energy inside the solid angle increases the incident light energy. Increasing incident light energy results in increasing reflected light energy. The constant of proportionality relating these two quantities, incident and reflected light, is the *bidirectional reflectance distribution function* (BRDF) and is usually represented as

$$\rho_{bd}(\mathbf{x}, \vec{\omega_i}, \vec{\omega_o}) = \frac{L_o(\mathbf{x}, \vec{\omega_o})}{L_i(\mathbf{x}, \vec{\omega_i}) \cos\theta_i \; d\omega_i} \tag{3.2}$$

where $\mathbf{x}$ is the point of incidence, $\vec{\omega_i}$ and $\vec{\omega_o}$ are the incoming and outgoing directions, and L_i and L_o are the incoming and outgoing radiances. Note, however, that the BRDF is not just the ratio of incoming and outgoing radiances; it also depends on the incoming projected solid angle $\cos\theta_i \;\; d{\omega_i}^3$. This dependency in the projected solid angle gives the units of inverse steradians $[{sr}^{-1}]$ to the BRDF and makes the BRDF to vary from zero to infinity. Interesting observations about and properties of the BRDF are:

- The BRDF is bidirecional because it depends on two directions—incoming and outgoing.
- The BRDF is a distribution function because it is always positive.
- The BRDF obeys the *reciprocity principle*: interchanging the incoming and outgoing directions does not change the BRDF value.
- The BRDF is, in general, anisotropic as described above.

Figure 3.5 illustrates BRDFs and BTDFs that capture surface roughness as discussed in Section 3.5.3.

[3] The solid angle subtended by an object is the surface area of its projection onto the unit sphere. The projected solid angle is the projection of that surface area onto the base of the sphere—the $theta = 0$ plane in spherical coordinates.

3.6 Light Transport in a Particular Direction

Section 3.5.3 described the scattering of a single incident ray of light into an entire reflection or transmission hemisphere. This section reverses that process; it considers the outgoing (reflected or transmitted) light in a single direction. By the same scattering argument above, the outgoing light in a single direction is composed of scattered light from all the directions in the incident hemisphere. Each of the incident rays in the hemisphere is scattered into the entire outgoing hemisphere and has a component in the particular outgoing direction of interest. All of these scattered components in the particular direction of interest add up to compose the final outgoing light. Note the linear behavior of light transfers assumed above.

Quantitatively, limiting the analysis to reflection, the reflected radiance L_o from a point $\mathbf{x}$ in direction $\vec{\omega_o}$ is given by the integral of the incoming radiance L_i over the entire hemisphere of incident directions, Ω_i, around $\mathbf{x}$, weighted by the corresponding BRDF ρ_{bd} and by the projected solid angle $\cos\theta_i \; d\omega_i$:

$$L_r(\mathbf{x}, \vec{\omega_o}) = \int_{\Omega_i} \rho_{bd}(\mathbf{x}, \vec{\omega_i}, \vec{\omega_o}) L_i(\mathbf{x}, \vec{\omega_i}) \cos\theta_i \; d\omega_i \tag{3.3}$$

The same equation applies for incoming transmitted light contributing for a single outgoing direction. The BRDF $\rho_{bd}(\mathbf{x}, \vec{\omega_i}, \vec{\omega_o})$ in Equation (3.3) is replaced by the corresponding *bidirectional transmittance distribution function* (BTDF) $\tau_{bd}(\mathbf{x}, \vec{\omega_i}, \vec{\omega_o})$ and the incoming hemisphere becomes the transmission hemisphere.

3.7 Light Transport Between Two Surfaces

So far, the interaction of light with matter has been discussed without considering the light's origin. In nature, this light could come directly from light sources or indirectly from reflections or transmissions with other surfaces in the environment. This section considers the possible light transfers between two surfaces, as introduced by Wallace [Wallace87]. The next section extends the concept to unlimited transfers along paths of light.

According to our description in Section 3.5.3, light can leave a surface diffusely or specularly. The main point here is with respect to the directionality of the energy emanation. Diffuse transfers are non-directional, whereas specular transfers imply directionality. By combining these diffuse and specular behaviors on the two ends of a two-surface light transfer, four distinct mechanisms are possible

[Wallace87], as illustrated in Figure 3.6: *diffuse-to-diffuse*, *specular-to-diffuse*, *diffuse-to-specular*, and *specular-to-specular*.

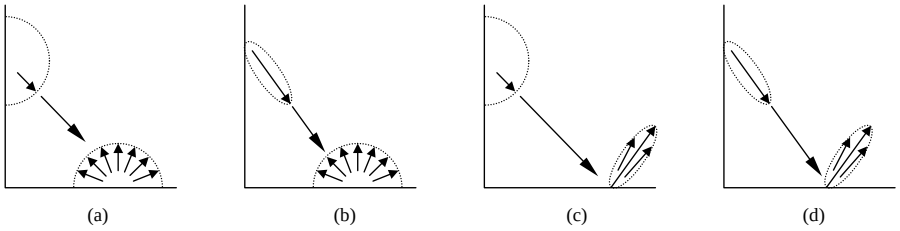


Figure 3.6: *Mechanism of light transfer between two surfaces: (a) diffuse-to-diffuse, (b) specular-to-diffuse, (c) diffuse-to-specular, and (d) specular-to-specular. (after [Wallace87])*

Note how the four mechanisms were organized in Figure 3.6. The first two ended on a diffuse transfer and the last two ended on a specular transfer. This particular organization suggests an interesting property: mechanisms ending with diffuse transfers are direction-independent, whereas mechanisms ending with specular transfers are direction-dependent. This property is particularly important when a viewer is introduced in the scene. Such a viewer can be a light sensor in photometry, a camera in photography and graphics, and a surface point in the light transport problem. The viewer perceives the same light intensity irrespective of its position and orientation with respect to diffuse transfers, whereas specular transfers produce varying light intensity as a function of the position of the viewer with respect to the surface generating the specular transfers. This concept will be key in the decomposition of the light transport problem and in preprocessing some of the resulting components in the next chapters.

3.8 A Complete Light Transport Model

The previous section discussed light transport between two surfaces: a point on a surface transfers light that originated from another surface in the scene. The full light transport problem involves the recursive application of this two-surface transport principle. The illumination at a point on a surface is determined by taking into account all the energy transferred from the rest of the environment into that point. However, the points on the rest of the environment also depend on the application of the same principle; they may receive light indirectly from other surfaces in the scene. A recursive approach is then necessary to solve the full light transport problem.

3.8.1 A Language for the Light Transport Problem

The recursive nature of the light transfer problem is commonly analyzed with transport paths represented using regular expressions [Heckbert91]. Energy transfers from the light sources through the environment to the observer's eye are represented with a word constructed from the alphabet $\{L, D, S, E\}$. L indicates emission from the light source, E represents absorption by the observer's eye, and D and S indicate diffuse and specular transfers (the specular component includes glossy behavior, as described in Section 3.5.3). The simplest light transport path is LE—light transport direct from the source into the observer's eye. Two other simple transport paths describe light leaving a source and getting to the observer's eye through a single diffuse or through a single specular transfer. The corresponding expressions are LDE and LSE. Additional symbols enable the description of more interesting paths but still preserve the compactness of the notation. The symbol '$|$' indicates the OR operation, and the symbols '$($' and '$)$' have the usual precedence meaning. For example, the local-illumination model usually implemented in graphics hardware is represented with $L(D|S)E$, which means that light reaches the observer's eye through diffuse or specular transfers but via a single bounce off the surface in question. The superscripts '*' and '$^+$' on a term indicate its repetition. The symbol '*' includes the null repetition of the term which is not allowed with '$^+$'. The symbol '$^+$' has the "at least once" meaning.

The full light transport problem is represented with $L(D|S)^*E$, which represents light reaching the eye through any number (including zero) of diffuse or specular transfers in no particular order. Note the differences between the light paths captured by the graphics hardware—$L(D|S)E$—and the light paths captured by the full light transport—$L(D|S)^*E$. For example, the graphics hardware is limited to light transfers through a single surface, whereas the full light transport problem may require recursive light transfers throughout the environment captured by the superscript '*'. More specifically, specular transfers in the graphics hardware are limited to highlights due to the light source on the reflective surface; the full light transport model can produce highlights on specular surfaces due to light sources or due to any other surface transferring light in the environment, as in nature.

In this dissertation, this light path notation will prove useful in characterizing and comparing different algorithms for approximating the full light transport problem.

3.9 The Light Transport Equation

Section 3.6 discussed the light transport for a single surface. Then, Section 3.7 described the light transport between two surfaces. Finally, Section 3.8 analyzed the full light transport problem with an indefinite number of interacting surfaces. Section 3.6 also presented an equation to determine the outgoing radiance from a single surface. This section extends the interpretation of that approach and presents the full light transport equation.

In nature, light leaving a point can be the result of emission, reflection, or transmission. The light transport equation combines these three components in an integral equation. In addition, the light transport equation captures the recursive nature of the light transport problem with an implicit equation. The *light transport equation* is an integral equation that describes the total radiance leaving a surface point in a given direction by taking into account all possible energy transfers in a scene:

$$\begin{aligned} L_o(\mathbf{x}, \vec{\omega_o}) \;\; = \;\; & L_e(\mathbf{x}, \vec{\omega_o}) + \\ & \int_{\Omega_r} \rho_{bd}(\mathbf{x}, \vec{\omega_o}, \vec{\omega_i})\; L_i(\mathbf{x}, \vec{\omega_i})\; cos\,\theta_i\; d\vec{\omega_i} + \\ & \int_{\Omega_t} \tau_{bd}(\mathbf{x}, \vec{\omega_o}, \vec{\omega_i})\; L_i(\mathbf{x}, \vec{\omega_i})\; cos\,\theta_i\; d\vec{\omega_i}. \end{aligned} \tag{3.4}$$

Figure 3.7 presents the notation for the light transport equation. The total outgoing radiance, $L_o(\mathbf{x}, \vec{\omega_o})$, leaving point $\mathbf{x}$ in outgoing direction $\vec{\omega_o}$, depends on a direct term and two indirect terms. The direct term, $L_e(\mathbf{x}, \vec{\omega_o})$, represents the radiance emitted from point $\mathbf{x}$ in direction $\vec{\omega_o}$. The indirect terms denote the reflected and the transmitted radiance from point $\mathbf{x}$ in direction $\vec{\omega_o}$. These terms depend on the integration of incoming radiance, $L_i(\mathbf{x}, \vec{\omega_i})$, over the reflection and transmission hemispheres, Ω_r and Ω_t, covering the surface at point $\mathbf{x}$. The integration is weighted by the bidirectional reflectance and transmittance distribution functions of the material, $\rho_{bd}(\mathbf{x}, \vec{\omega_o}, \vec{\omega_i})$ and $\tau_{bd}(\mathbf{x}, \vec{\omega_o}, \vec{\omega_i})$, relating the outgoing and incoming directions at the point and at the direction in question. The term $\cos\theta_i\, d\vec{\omega_i}$ represents the projected solid angle, as described in Section 3.5.5.

Note that a single application of the light transport equation captures what a viewer sees from a single light transfer, as described in Section 3.6. The light transport between two surfaces is determined with two successive applications of the light transport equation. The complete light paths of the full light transport problem are captured with the implicit integrals and require recursive applications of the light transport equation.

Equation (3.4), in a slightly different form, was first introduced to computer graphics by Kajiya [Kajiya86] in 1986, who named it *rendering equation*. His approach was based on *two-point transport*

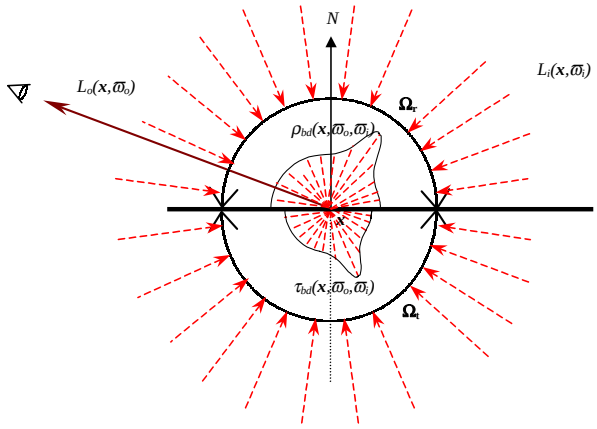


Figure 3.7: *Notation for the radiance equation.*

of energy[4], instead of directional or solid angle transport as used in this dissertation. Glassner [Glassner95] refers to Equation (3.4) as the *radiance equation*, as it defines the outgoing radiance from a point on a surface in a given direction. Sillion and Puech [Sillion94] refer to the same equation as the *global illumination equation*. Although all the names above are appropriate for Equation (3.4), we refer to that same equation as *light transport equation*, as this name correctly describes the particular problem we are trying to solve. We let the rendering, radiance, and global illumination characters be defined by the context of the discussions.

[4] The two-point energy transport depends on the position of two surface points connected by a straight line and does not involve solid angles.

CHAPTER 4

LINEAR DECOMPOSITION OF LIGHT TRANSPORT

Chapter 3 discussed the light transport problem and several assumptions that helped reduce the complexity of the problem. Under those simplifying assumptions, a light transport equation was presented for computing the radiance leaving a point on a surface in a scene in a particular outgoing direction. By solving that light transport equation, photorealistic rendering aims at producing synthetic images of computer modeled environments that approximate what an observer or a photographic camera would perceive in the corresponding real environment.

Conceptually, for static environments, interactive photorealistic rendering could precompute the entire light transport for a scene and then just query the final illumination or radiance function for each new pose for which a synthetic image is desired. Because many light paths in a scene are directionally dependent, precomputing the entire light transport demands very finely sampling a five-dimensional spatial domain. Also, although the full light transport is a five-dimensional spatial problem, interactive photorealistic rendering usually explores a very small subset of such domain, so computing the whole domain is very wasteful. Moreover, precomputing and storing all the radiance information of a scene is not computationally practical.

Alternatively, the full light transport problem can be broken down into smaller problems with simpler solutions which are then recombined to approximate the full solution. This chapter discusses a linear decomposition of the light transport equation, and then the next chapters describe a set of techniques for providing, in real time, approximate solutions to parts of that equation for reasonably complex models.

As seen in Section 3.9, the light transport equation is an implicit integral equation: the unknown radiance depends on the integration of itself; the unknown radiance appears both on the left-hand side and on the right-hand side of the equation. For that reason, the light transport equation is hard to solve. The computation of the outgoing radiance from a single point in a scene requires knowing the incoming

radiance from all directions around that point. The incoming radiance in a particular direction to the point of interest is the outgoing radiance from another point in the environment, which in turn depends on the incoming radiance from all directions around that other point. This describes the recursive process formally represented by the implicit equation.

Section 3.9 discussed the light transport equation explicitly in terms of emission, reflection, and transmission. This section, for clarity, combines the reflection and transmission terms into a single term. Formally, this reduces the light transport equation to (same notation as for Equation 3.4)

$$L_o(\mathbf{x}, \vec{\omega_o}) = L_e(\mathbf{x}, \vec{\omega_o}) + \int_{\Omega} f_{bd}(\mathbf{x}, \vec{\omega_o}, \vec{\omega_i})\ L_i(\mathbf{x}, \vec{\omega_i})\ cos\ \theta_i\ d\vec{\omega_i} \tag{4.1}$$

where Ω is the complete sphere of directions around point $\mathbf{x}$ composed by the union of the hemispheres Ω_r and Ω_t, and $f_{bd}(\mathbf{x}, \vec{\omega_o}, \vec{\omega_i})$ is the respective bidirectional distribution function (combined BRDF and BTDF). The first additive component of Equation (4.1) represents the direct light emitted by point $\mathbf{x}$, and the second term represents the indirect light reflected and/or transmitted by point $\mathbf{x}$.

Given the distinct characteristics of different parts of the light transport problem, it is possible to decompose the problem into simpler terms. The idea is to rephrase the complex light transport problem into simpler problems that can benefit from particular strengths of different solution methods. At run time, those components are recombined to approximate a solution of the light transport equation and to produce synthetic images. To enable exploitation of current graphics hardware when recombining the light transport components in real time, the decomposition needs to be compatible with the linear operations available in hardware.

Let's start by going back to the original light transport Equation (3.4) from Section 3.9. Abstractly, that equation states that the perceived radiance (light) is given by

$$perceived\ =\ emitted\ +\ reflected\ +\ transmitted. \tag{4.2}$$

We combine the reflected and transmitted terms into a single component and characterize the light transport problem as

$$perceived\ =\ emitted\ +\ transferred. \tag{4.3}$$

The basic distinction between the two terms is the path of light from the source to the viewer. In the language for representing light paths discussed in Section 3.8.1, the equation above can be written as

$$perceived\ =\ LE\ +\ L(D|S)^{+}E. \tag{4.4}$$

The first term is trivial, as it represents a single light transfer—the case where the point in question is a light source. That is, the transfer of light from a light source (L) to a viewer (E). The second term is more elaborate and represents all the possible transfers that light may undergo from light sources until it reaches a viewer, ranging from ideally diffuse to ideally specular in reflection and/or transmission transfers.

We can now split each of the terms of Equation (4.4) according to the directional distribution of the outgoing light transfer. We split each of the terms into two categories: the light paths that end [1] with an ideally diffuse transfer, and the light paths that end with a specular (or non-diffuse) transfer. In our simplified equation this means:

$$perceived \;=\; L_D E \;+\; L_S E \;+\; L(D|S)^* DE \;+\; L(D|S)^* SE. \tag{4.5}$$

where the subscripts $_D$ and $_S$ in the first two terms represent light sources with diffuse and specular distribution properties. (Remember that we defined any non-directionally-uniform distribution as *specular* in Section 3.5.3.) Note that the light paths in the indirect terms (last two terms) can have any combination of diffuse and specular transfers until they reach the last transfer before reaching the observer; the restriction is only with respect to the last light transfer before it reaches the viewer. The last light transfers before reaching the viewer define the ultimate behavior light that the viewer sees from a light path. The major point here is that once light reaches an ideally diffuse surface it becomes diffusely distributed—it loses any directionality present along the light path from specular transfers. With respect to an arbitrary viewer in the scene, the intensity of diffusely distributed light is direction-invariant—it is called *view-independent*. This means that we can solve the light transport problem for all the ideally diffuse surfaces (third term in Equation (4.5)) in a scene by considering how they recursively "see" the entire environment (including how they "see" specular surfaces). The light intensity of specularly distributed light is not directionally invariant—it is called *view-dependent*—which implies that the light transport problem for non-diffuse surfaces (last term in Equation (4.5)) is less pre-computable. Chapter 5 details the view-independent component and Chapters 6 and 7 discuss the view-dependent component.

[1] A light path starts at a light source and ends at a viewer, but, in terms of light transfers, a light path ends with a transfer from a surface to a viewer.

4.1 Casting the Light Transport Equation in Terms of Convolution

This section introduces a new and simplified alternative form for representing integral equations in terms of the convolution operation. Regarding the solution of integral equations, numerical methods are normally used, as analytical solutions are frequently impractical. Glassner [Glassner95] presents a good overview of numerical methods for solving integral equations. This section presents a new alternative approach for solving integral equations, following a "signals and systems" approach in terms of convolution [Oppenheim96]. The advantage of this approach, in the context of image synthesis, is the possibility of evaluating the light transport equation in real time by exploiting the hardware-assisted convolution available in current graphics architectures.

Consider the problem of light interacting with matter, as discussed in Chapter 3. The central discussion was how light responds to materials, i.e., how light shining on a material gets changed by the material and scattered to produce the outgoing light. This section shows how that same problem can be described in terms of signals and systems. In the terminology of Chapter 2, the light is our signal and the materials are the systems. The analogy is as follows. Incoming light impinging on a material represents an input signal to a system. The material is the system. Remember, from Chapter 2, that a system is a black box and that we are only interested in the output signal produced by the system for a given input, without considering the process that changes the input signal to produce the output signal. The outgoing light (emitted, reflected, and/or transmitted) represents the output signal. Also from Chapter 2, a system obeys certain properties and relates an input signal to its corresponding output signal through the convolution operation. Every system has an impulse response, and the output signal of a system is the convolution of the input signal with the system's impulse response. We need to identify the impulse responses of materials, and we need to analyze the light transport problem in terms of convolution.

Formally, start by comparing the convolution equation

$$g(x) = f(x) \star k(x) = \int_{-\infty}^{+\infty} f(u)\ k(x-u)\, du$$

(Equation (2.7)) and the light transport equation

$$L_o(\mathbf{x}, \vec{\omega_o}) = L_e(\mathbf{x}, \vec{\omega_o}) + \int_{\Omega} f_{bd}(\mathbf{x}, \vec{\omega_o}, \vec{\omega_i})\ L_i(\mathbf{x}, \vec{\omega_i})\ cos\ \theta_i\ d\vec{\omega_i}.$$

(Equation (4.1)). There is a clear formal similarity between the convolution equation and the second term of the light transport equation; both integrate the product of two functions. This suggests that the light transport equation can be expressed in terms of convolution.

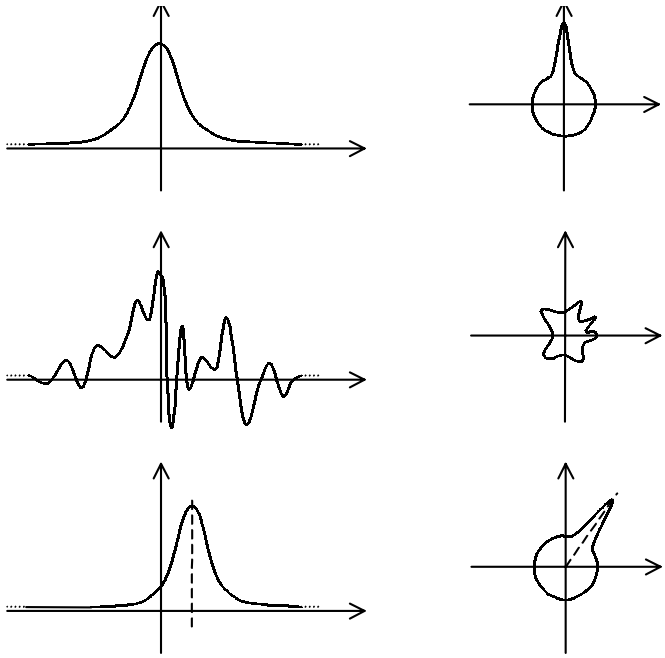


Figure 4.1: *Convolution kernel and signal in a rectilinear infinite space (left) and in a directional space (right): the first row shows the convolution kernel defined at its origin; the second row shows a signal in the same space as the kernel, and the last row shows the kernel shifted to (centered at) a particular point/direction of interest for the infinite/directional space.*

However, there is a necessary mental shift for understanding this new representation for the light transport equation in terms of convolution. The reader should note that a convolution kernel is a function defined in the same space as its corresponding signal (see Figure 4.1). The convolution equation also implies that, for each point where we want to evaluate a convolution, the kernel origin is shifted to that evaluation point (by the subtraction in the independent variable of the kernel); the kernel is said to be *centered* at the evaluation point. Additionally, it is necessary to define a new space for convolution. Instead of defining convolution in the traditional rectilinear space with an infinite range, the light transfer problem is carried out in a directional space[2] around the point of interest as seen in Figure 4.1(right), (Section 2.3.1). Both the signal and the kernel are defined in this directional space in terms of directions (or angles in spherical coordinates). Centering a kernel in this new space means shifting the kernel from its origin to a particular given direction.

[2] A *directional space* is defined by all the possible directions emanating from a point.

To get the light transport equation closer in form to the convolution equation, let's create two auxiliary functions:

$$F(\mathbf{x}, \vec{\omega_i}) = L_i(\mathbf{x}, \vec{\omega_i}) \tag{4.6}$$

$$K(\mathbf{x}, \vec{\omega_{io}}, \vec{\omega_i}) = f_{bd}(\mathbf{x}, \vec{\omega_o}, \vec{\omega_{io}} - \vec{\omega_i}) \cos\theta_i \tag{4.7}$$

where $F(\mathbf{x}, \vec{\omega_i})$ is the incoming radiance at point $\mathbf{x}$ and at direction $\vec{\omega_i}$, and $K(\mathbf{x}, \vec{\omega_{io}}, \vec{\omega_i})$ is a re-parameterization of the bidirectional distribution function $f_{bd}(\mathbf{x}, \vec{\omega_o}, \vec{\omega_i})$ centered around a direction $\vec{\omega_{io}}$. The incoming direction for centering the convolution kernel in directional space, $\vec{\omega_{io}}$, is either the ideally reflected direction or the ideally transmitted direction (see sections 3.5.1 and 3.5.2) derived from the desired outgoing direction $\vec{\omega_o}$ (Figure 4.2). That is, $\vec{\omega_{io}}$ defines an axis in the space of all directions emanating from point $\mathbf{x}$ and the convolution kernel K is centered around the direction of that axis. Figure 4.2 illustrates the kernel from Figure 4.1(right) centered at point $\mathbf{x}$ and about direction $\vec{\omega_{io}}$.

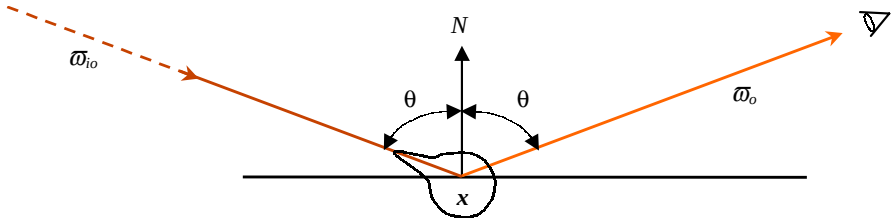


Figure 4.2: *Centering the kernel from Figure 4.1(right) about direction $\vec{\omega_{io}}$ in directional space.*

Note that the kernel in the traditional convolution Equation (2.7) is centered at the desired point $\mathbf{x}$ for which we want to evaluate the convolution in that space. In terms of the light transport equation, there is an outgoing direction in directional space for which we want to evaluate the convolution. However, the kernel is not centered at that direction. Instead, the kernel is centered at a direction that is a function of the desired outgoing direction. There are two reasons for choosing the ideally transferred direction to be the axis for centering the BDF convolution kernel, instead of any other direction in the hemisphere. First, the ideally transferred direction is usually the direction or close to the direction of greatest density of energy in any light transfer, which corresponds to the peak of the kernel in the convolution-based approach. Secondly, the spread of contributions of incoming light for a single desired outgoing direction depends, in general, on a neighborhood around the ideally transferred direction, which defines the kernel support in the convolution-based approach. An exception to the

first rule is retro-reflection[3], because it requires an alignment of the viewer with the light source for the effect to be observed. To take retro-reflection into account in our convolution-based model, the kernel needs to be centered at the viewing direction, instead of being centered at the ideally reflected direction derived from the viewing/outgoing direction.

Note that the integration variable in the convolution Equation (2.7) controls the displacement around the central desired point and that the integration limits indirectly represent the kernel support (how far in the domain the convolution should gather information about the signal, Section 2.6.2). Analogously, in our convolution- based view of the light transport equation, the integration variable controls the angular displacement around a central incoming direction of interest and the integration limits represent the size of the solid angle swept out around this central direction, i.e., define the kernel support. In directional space, the kernel support can be as wide as the complete corresponding hemisphere (reflection or transmission hemisphere) in question. In practice, the kernel support is determined by the solid angle subtending the incoming nonzero BDF lobe for the corresponding desired outgoing direction. Figure 4.3 presents radiance integration over a solid angle subtending a BRDF lobe around direction $\vec{\omega_{io}}$. Notice the difference between the kernels (BRDFs) in figures 4.2 and 4.3—the kernel in Figure 4.3 has a finite support, whereas the kernel in Figure 4.2 does not.

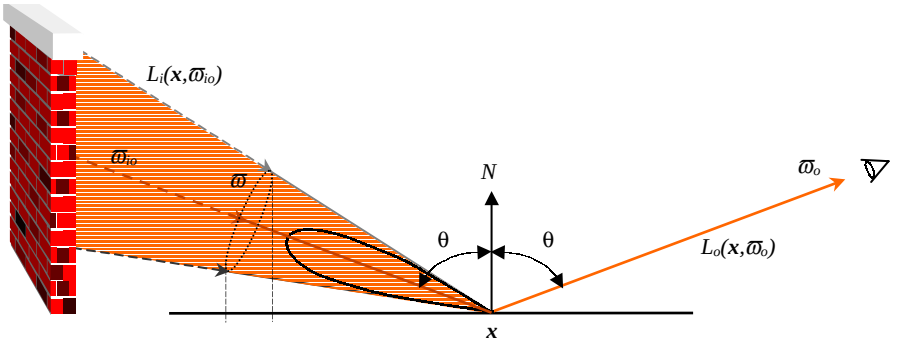


Figure 4.3: *Radiance integration for a small solid angle subtending a BRDF lobe around direction* $\vec{\omega_{io}}$.

Function K represents the impulse response (or the point spread function) of the surface material to the given incoming light. It describes how incoming light from a single direction $\vec{\omega_{io}}$ is spread into the hemispheres of outgoing reflected/transmitted directions. Alternatively, given the reciprocity property of the distribution functions, the point spread function K can be seen as describing how incoming hemispherical light is integrated (by reflection or by transmission) into a single outgoing

[3] *Retro-reflection* refers to the effect which produces reflected light around the direction anti-parallel to the incoming direction.

direction $\vec{\omega_o}$. That is, given an outgoing direction, the PSF describes how the incoming (or input) light in the corresponding hemisphere gets weighted and integrated to produce the output light in that direction. The PSF weights the contributions of incoming light over the neighborhood of directions (solid angle) covered by the kernel, i.e., in the kernel support.

The radiance Equation (4.1) can then be expressed in a simpler form in terms of a convolution operation in directional space:

$$L_o(\mathbf{x}, \vec{\omega_o}) = L_e(\mathbf{x}, \vec{\omega_o}) + F(\mathbf{x}, \vec{\omega_i}) \star K(\mathbf{x}, \vec{\omega_{io}}, \vec{\omega_i}) \tag{4.8}$$

where F and K are as described above. This notation hides the space in which convolution takes place. Although the convolution Equation (2.7) defines that integration is carried out in an infinite space, notice that integration in the light transport Equation (4.1) is performed in the two complementary hemispheres of directions (reflection and transmission hemispheres) covering the point $\mathbf{x}$ in question. To accomplish the same integration as performed by the radiance Equation (4.1), this implies that convolution in Equation (4.8) is computed in directional/hemispherical space, as discussed in Section 2.3.1. In practice, integration is usually performed over only a subset of the hemisphere—the subset that subtends the nonzero values of the BDF (Figure 4.3).

Chapter 3 discussed different types of materials in terms of bidirectional distribution functions. We can now discuss the different types of convolution kernels required by different surface materials (Figure 4.4):

- Diffuse transfers require convolution with a constant kernel over the entire hemisphere—radiance leaving point $\mathbf{x}$ in a single direction is given by integrating radiance from all the directions in the hemisphere weighted by a directionally constant value (the diffuse reflectance at $\mathbf{x}$);

- Mirror-like transfers require convolution with a delta kernel over a single direction (or no convolution at all)—radiance leaving point $\mathbf{x}$ in a single direction is given by radiance from a single incoming direction;

- Glossy transfers require convolution with non-constant kernels over a solid angle subtending the corresponding glossy BRDF lobe—radiance leaving point $\mathbf{x}$ in a single direction is given by radiance integration from all the directions in the hemisphere weighted by directional reflectance/transmittance values.

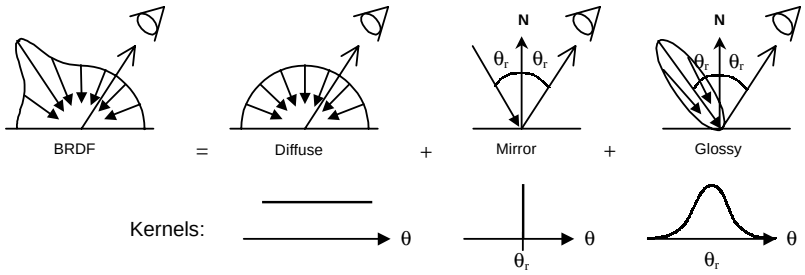


Figure 4.4: *Decomposing a BRDF into qualitative components and its corresponding kernels in one-dimensional directional space.*

The most general glossy transfers also require spatially varying kernels, as discussed in Section 2.3.2. Both the shape and the amplitude of the kernels may vary for such materials. Figure 4.5 illustrates the need for a spatially varying kernel in general glossy transfers. Two orientations of the viewer with respect to the surface are shown. Besides geometry, the figure also presents the corresponding kernels—the one-dimensional magnitude plot of the BRDF for the given situations. Notice the change in shape and magnitude of the kernel for the non-grazing and grazing situations. This means that the kernel is view-dependent; a different kernel is needed for each outgoing direction.

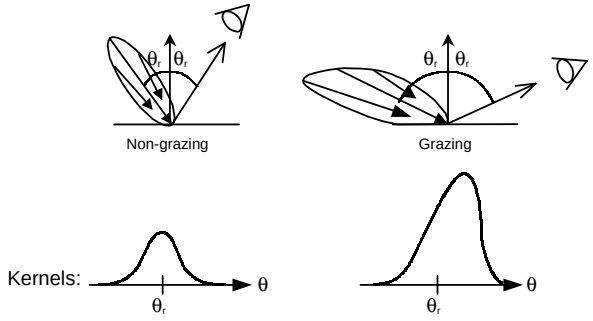


Figure 4.5: *Spatially varying kernel for glossy reflection at non-grazing and grazing situations on a one- dimensional directional space.*

4.2 Decomposing the Light Transport Equation

This section discusses a more formal decomposition of the light transport equation than the one presented in the beginning of this chapter. We rewrite the qualitatively decomposed Equations (4.3) and (4.5) formally. We also present the equation which will be approximated by our rendering methods discussed in the next chapters.

The emitted term in Equation (4.3) was split into two terms in Equation (4.5)—diffuse emission and specular emission. Since diffuse emission is independent of the outgoing direction, we write this term only in terms of the location **x** where the light transfer takes place—$L_{D_e}(\mathbf{x})$. Specular emission, however, depends on the outgoing direction of light and we write it as $L_{S_e}(\mathbf{x}, \vec{\omega_o})$. The emitted term is then

$$emitted = L_{D_e}(\mathbf{x}) + L_{S_e}(\mathbf{x}, \vec{\omega_o}). \tag{4.9}$$

Also from Equations (4.3) and (4.5), the transferred term was split into two terms—the ideally diffuse component and the non-diffuse component. This splitting of the transferred term is based on a common treatment of general BRDFs, which are frequently represented as the sum (superposition) of three qualitative components [Cohen93] (Figure 4.4(top)): diffuse reflection, mirror reflection, and glossy reflection. According to our definition in Chapter 3, we can combine mirror reflections and glossy reflections into specular reflections and write a bidirectional distribution function f_{bd} as the sum of the diffuse component and the specular component:

$$f_{bd}(\mathbf{x}, \vec{\omega_o}, \vec{\omega_i}) = \rho_D(\mathbf{x}) + \rho_S(\mathbf{x}, \vec{\omega_o}, \vec{\omega_i}). \tag{4.10}$$

By substituting this decomposition of the BRDF into the transferred term of the light transport Equation (4.1) we get:

$$transferred = \int_\Omega [\rho_D(\mathbf{x}) + \rho_S(\mathbf{x}, \vec{\omega_o}, \vec{\omega_i})] L_i(\mathbf{x}, \vec{\omega_i})\ cos\ \theta_i\ d\vec{\omega_i}, \tag{4.11}$$

which can be split into two integrals

$$transferred = \int_\Omega \rho_D(\mathbf{x}) L_i(\mathbf{x}, \vec{\omega_i})\ cos\ \theta_i\ d\vec{\omega_i} + \int_\Omega \rho_S(\mathbf{x}, \vec{\omega_o}, \vec{\omega_i}) L_i(\mathbf{x}, \vec{\omega_i})\ cos\ \theta_i\ d\vec{\omega_i}. \tag{4.12}$$

We can now factor $\rho_D(\mathbf{x})$ out of the integral in the diffuse or view-independent term, since $\rho_D(\mathbf{x})$ is independent of the integration variable:

$$transferred = \rho_D(\mathbf{x}) \int_\Omega L_i(\mathbf{x}, \vec{\omega_i})\ cos\ \theta_i\ d\vec{\omega_i} + \int_\Omega \rho_S(\mathbf{x}, \vec{\omega_o}, \vec{\omega_i}) L_i(\mathbf{x}, \vec{\omega_i})\ cos\ \theta_i\ d\vec{\omega_i}. \tag{4.13}$$

Finally, we rewrite the specular or view-dependent component in terms of our convolution form of the light transport equation (Section 4.1) and get

$$transferred = \rho_D(\mathbf{x}) \int_\Omega L_i(\mathbf{x}, \vec{\omega_i})\ cos\ \theta_i\ d\vec{\omega_i} + L_i(\mathbf{x}, \vec{\omega_i}) \star K_s(\mathbf{x}, \vec{\omega_{io}}, \vec{\omega_i}) \tag{4.14}$$

where K_s represents the convolution kernel for the specular components—mirror and glossy—of the BRDF as described in the previous section.

To conclude, we combine the emitted and the transferred terms into our decomposed light transport equation:

$$L_o(\mathbf{x}, \vec{\omega_o}) = L_{D_e}(\mathbf{x}) \; + \; L_{S_e}(\mathbf{x}, \vec{\omega_o}) \; + \; \rho_D(\mathbf{x}) \int_\Omega L_i(\mathbf{x}, \vec{\omega_i}) \; cos\, \theta_i \; d\vec{\omega_i} \; + \; L_i(\mathbf{x}, \vec{\omega_i}) \star K_s(\mathbf{x}, \vec{\omega_{io}}, \vec{\omega_i}), \tag{4.15}$$

which represents the multipass equation that we want to solve per pixel and per frame in real time. The next three chapters address the view- independent and view-dependent components separately.

CHAPTER 5

THE VIEW-INDEPENDENT COMPONENT

Chapter 4 considered a linear decomposition of the light transport equation in which one of the components was completely independent of the viewer in the environment. Light paths of ideally diffuse transfers capture effects such as diffuse highlights, shadows, and color bleeding[1], all of which are view-independent. This chapter discusses in more detail this view-independent component, a method for solving it, and our results for displaying that component smoothly in real time by exploiting features of current graphics hardware.

5.1 The Radiosity Equation

The light transport Equation (4.1) is greatly simplified under the assumption that all surfaces and light sources in a scene are ideally diffuse. This is usually called the *radiosity assumption* [Cohen93] and represents the view-independent component. The radiosity assumption implies that the BRDF of all surfaces in a scene is independent of the direction of incoming and outgoing directions (Section 3.5.3). In terms of the light transport equation, this assumption permits factoring the BRDF $\rho_{bd}(\mathbf{x})$ out of the integral, as it does not depend on the integration variable:

$$L_o(\mathbf{x}) = L_e(\mathbf{x}) \; + \; \frac{\rho_{bd}(\mathbf{x})}{\pi} \int_{\Omega} L_i(\mathbf{x}, \vec{\omega_i}) \; cos\, \theta_i \; d\vec{\omega_i}. \tag{5.1}$$

Note that, because the right-hand side of the equation is a non-directional quantity, there is no dependence on the outgoing direction $\vec{\omega_o}$ in Equation (5.1). The outgoing radiance from an ideally diffuse light transfer is invariant with respect to the relative orientation of the viewer and the ideally diffuse surface.

[1] *Color bleeding* refers to how the appearance of an object can be affected by the outgoing light from another colored object. For example, the appearance of a white ceiling can be affected by an adjacent red wall. This effect happens due to diffuse interreflections of light.

Alternatively to the radiance representation of the view-independent component, Equation (5.1) can be presented in terms of radiosity[2]. Divide Equation (5.1) by π to convert radiance into radiosity and remove the directional dependence to formulate the *continuous radiosity equation* in directional space:

$$B(\mathbf{x}) = E(\mathbf{x}) + \rho_d(\mathbf{x}) \int_{\Omega} B_i(\mathbf{x})\ cos\ \theta_i\ d\vec{\omega_i} \tag{5.2}$$

where $\rho_d(\mathbf{x}) = \frac{\rho_{bd}(\mathbf{x},\vec{\omega_o},\vec{\omega_i})}{\pi}$ is the constant reflectance at point $\mathbf{x}$, $E(x)$ is the emitted energy per unit area at point $\mathbf{x}$, and $cos\ \theta_i d\vec{\omega_i}$ is the projected solid angle ranging over the reflection hemisphere Ω_r.

Using the methods developed in Section 4.1, Equation (5.2) can be reformulated in terms of convolution as

$$B(\mathbf{x}) = E(\mathbf{x}) + \rho_d(\mathbf{x})\ [B_i(\mathbf{x}) \star cos\ \theta_i]\ . \tag{5.3}$$

However, there are simpler and more efficient ways for solving the radiosity problem. Because radiosity is a view-independent quantity, it can be completely precomputed and stored for later reuse at rendering time. The next section describes important features of the radiosity method relevant to this dissertation.

5.2 The Radiosity Method

Equation (5.2), in a slightly modified form, has been widely studied in thermal engineering since the 1950s as the problem of radiative transfer [Hottel54], and in computer graphics since the 1980s under the problem of ideally diffuse light transfer (radiosity) [Goral84, Nishita85]. There are complete books discussing the topic [Cohen93, Sillion94, Glassner95], and this section reviews only some fundamental ideas of the radiosity method relevant to this dissertation.

[2] *Radiosity*, represented by B, is the total energy leaving a surface per unit area. In the case of ideally diffuse environments, radiance is independent of incoming and outgoing directions; and radiosity is given by the integration of the outgoing radiance over the hemisphere of directions:

$$\begin{aligned} B(x) &= \int_{\Omega_r} L_o(\mathbf{x})\ cos\ \theta\ d\vec{\omega} \\ &= L_o(\mathbf{x}) \int_{\Omega_r} cos\ \theta\ d\vec{\omega} \\ &= \pi\ L_o(x). \end{aligned}$$

Notice that radiosity is directly proportional to outgoing radiance with a factor of π.

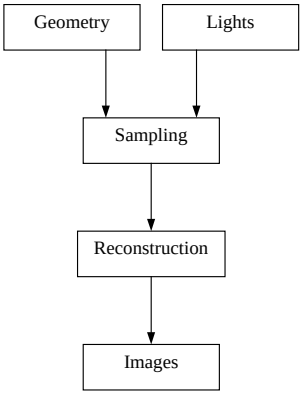


Figure 5.1: *Basic radiosity diagram.*

Figure 5.1 shows a high level diagram of the basic steps in the radiosity method which help connect the radiosity problem to the signals-and-systems problem of Chapter 2. The basic radiosity approach starts with a geometric model of an environment and information about the light sources. Then the environment is sampled to compute the radiosity function at discrete points. The approach concludes with a reconstruction step for image synthesis purposes. The next sections discuss and detail the sampling and reconstruction steps, and introduce our methods for displaying radiosity results smoothly in real time. The solution methods for radiosity, based on setting up and numerically solving a linear system of equations, are not central to this discussion because of the availability of a commercial product—*Lightscape*^{TM} from Discreet Logic—for computing radiosity. However, the sampling step is described in detail, since it affects our reconstruction techniques.

The continuous radiosity Equation (5.2), like the light transport equation, is rarely analytically solvable; numerical methods are necessary. The basic idea is to partition the surfaces of an environment into a finite number of small elements and to find a solution for a discrete version of the radiosity equation for those elements. Formally, the radiosity method can be derived in terms of finite-element theory. Instead of re-deriving the entire formalism, this chapter provides a more informal and heuristic approach to radiosity. The interested reader is referred to [Cohen93, Sillion94, Glassner95] for more formal approaches.

In radiosity, the domain is the union of the surfaces of all the objects in the scene and the radiosity function operates on that domain. For discretizing and computing the radiosity function of a scene, the domain (the surfaces of the scene) is subdivided into a *mesh of elements*. For computation, each of the resulting small surface elements is assumed to have a uniform (constant) radiosity value. The composition of all these constant radiosity elements approximates the radiosity function over the entire

domain. That is, the radiosity method, in general, approximates the radiosity function with a piecewise-constant function. In practice, the surface elements are usually represented with convex polygons and the final radiosity solution is represented with a mesh of quadrilaterals and/or triangles, each with its own color (Figure 5.2).

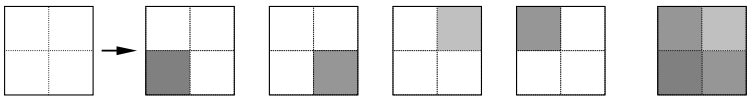


Figure 5.2: *Subdivision of a quadrilateral into* 2×2 *elements and the corresponding constant radiosity (coded in gray) for each element.*

Since radiosity deals only with ideally diffuse surfaces, the radiosity function of a scene contains mostly low-spatial-frequency details, as opposed to the sharp highlights possible in specular transfers. Although features like shadows can introduce higher frequencies at the edges between lit and shadowed regions, the general lighting of an ideally diffuse environment tends to have smoothly varying transitions across the surface domain, i.e., low-spatial-frequency contents. This observation about the spatial frequency content of the radiosity functions can be exploited in sampling and reconstruction. Samples can be placed far apart in regions of low frequency content and closer in regions with higher frequency content. This discretization of the radiosity function can then be reconstructed with a low order polynomial. The next sections discuss the sampling and reconstruction issues involved in the radiosity method.

5.3 Sampling the Radiosity Function

The scene discretization into surface elements for radiosity computation is a sampling task that can lead to artifacts in the final result. Large elements in regions with high radiosity gradients can result in missing features in the reconstructed radiosity function. The simple brute-force approach of just using finer uniform meshes rapidly increases storage and computational time costs, and may still be insufficient for capturing small features. The domain needs to be finely subdivided only where it will significantly improve the accuracy of the reconstructed radiosity.

Obtaining an optimal mesh requires knowledge about the radiosity function. Meshing techniques that use such information can be characterized as either *a priori* or *a posteriori*. *A priori* meshing techniques are used to determine features dependent only on the geometry of the environment and on the light sources, such as direct or primary shadows, before the radiosity solution is computed.

After the solution has been partially computed, *a posteriori* meshing techniques are then used to refine the mesh in regions with high radiosity gradients. *A priori* methods are beyond the scope of this dissertation and the reader is referred to [Lischinski92, Heckbert92] for good discussions on the topic.

A posteriori meshing refines the mesh in a recursive adaptive-subdivision manner, after the radiosity solution has been partially completed. Initially, a radiosity approximation is obtained using a mesh determined *a priori*—usually, the mesh of polygons describing the original scene. That mesh is then refined in regions where the local error is potentially high (e.g., regions with high radiosity gradient), using the initial approximation of the radiosity function.

Radiosity is usually sampled in a regular form on the surfaces of a scene (see Figure 5.3). Each original surface in a scene is normally either of quadrilateral or triangular shape. The root

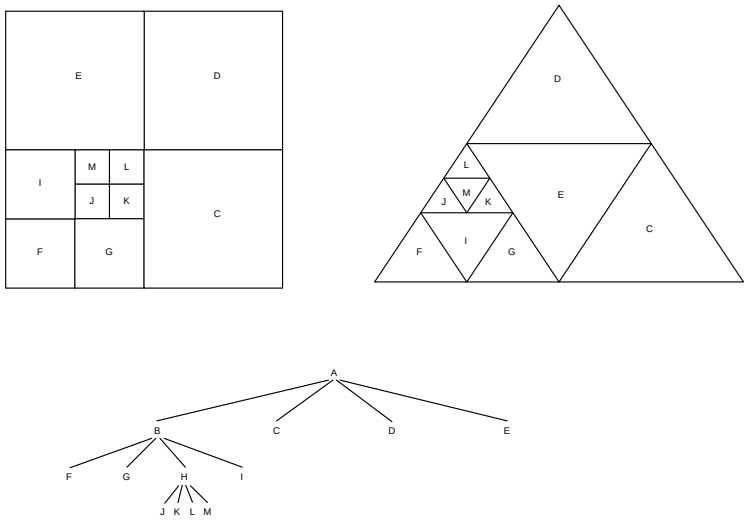


Figure 5.3: *Adaptive subdivision of a quadrilateral and a triangle and the corresponding quadtree.*

of a quadtree[3] is associated with each original surface. The corresponding quadtree then spatially partitions the surface hierarchically into smaller elements. Nodes of the tree store radiosity at the respective hierarchical level, representing the radiosity associated with the corresponding region of the original surface. Note that both quadrilaterals and triangles are partitioned into four elements, since this partitioning preserves the aspect ratio of the original unsubdivided primitive and unifies the quadtree approach (notice in Figure 5.3 that both the particular quadrilateral partitioning and triangle partitioning are represented by the same quadtree).

[3] A quadtree is a two-dimensional binary tree in which each node has zero or four children.

The standard way of computing, storing, and displaying radiosity derives a mesh of polygons from each quadtree (surface) in the scene. A traversal of a quadtree, followed by the corresponding partitioning of the associated original geometry, creates polygons at leaf nodes of the quadtree. Note how the compositing of all the leaf nodes of the quadtree of Figure 5.3 represents the original unsubdivided polygons. Remember also that each of those nodes contains a uniform radiosity value sampled from the radiosity function at that region of the domain.

5.4 Reconstructing the Radiosity Function

The discretized radiosity function obtained in the previous section represents a discrete sampling of the radiosity function at selected regions of a scene. The process of synthesizing images of such a scene is to reconstruct a function that approximates the radiosity function in the entire environment, i.e., a function that preserves the spatial discontinuities of the radiosity function but is otherwise smooth. That reconstructed or interpolated function is usually called a *reconstruction radiosity function*.

Assume that the radiosity function was evaluated (numerically approximated) at points V of the domain. The reconstruction problem aims to find a reconstruction function $\mathcal{R}$ that, at any point $v \in V$ of the domain, provides the exact same values as the corresponding radiosity samples, and approximates the radiosity function $\mathcal{B}$ everywhere else in the entire domain:

$$\mathcal{R}(v) = \mathcal{B}(v),\ \forall v \in V \tag{5.4}$$

$$\mathcal{R}(p) \cong \mathcal{B}(p),\ \forall p \notin V. \tag{5.5}$$

The same reconstruction function should approximate the first derivative, or gradient, of the radiosity function on the entire domain and should also preserve the continuity and discontinuities of the radiosity function. This formalism is useful in comparing synthetic images produced from radiositized models. Each pixel p of such an image approximates the actual radiosity function of the scene such that $\mathcal{R}(p) \cong \mathcal{B}(p)$.

By analyzing the illumination of a real scene one can observe the type of continuity and discontinuities expected from a reconstructed radiosity function. Consider an empty cubic room with six gray faces and a single light source on the ceiling (Figure 5.4). Simple inspection shows that the light distribution in such a room is continuous on the walls, on the ceiling, and on the floor. However, observe the radiosity value discontinuities in illumination distribution at the edges between the orthogonal faces—any two non-coplanar faces sharing an edge may cause value discontinuities at

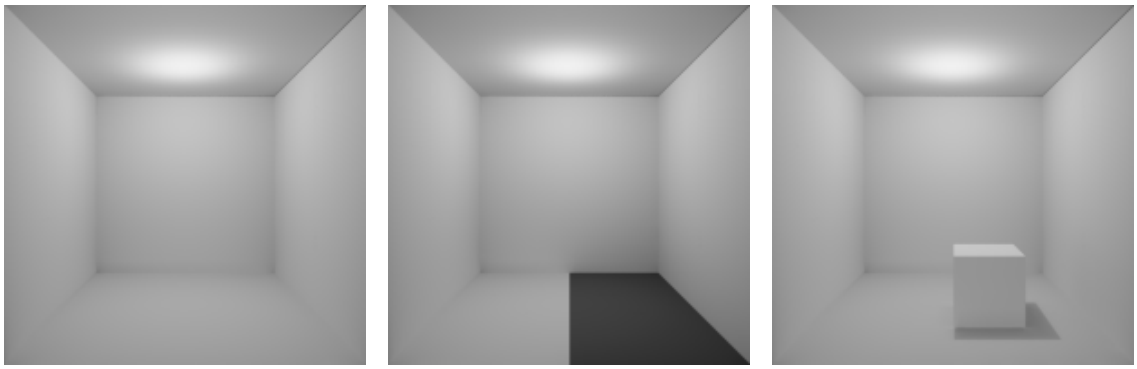


Figure 5.4: *A cubic gray room illuminated by a point light source near the ceiling. Notice that the light reflected off the ceiling acts like an area light source for the scene. Notice also the continuity and discontinuities of the illumination function across the domain.*

the edge. Additionally, consider that the floor is now split in two coplanar materials: half is still of the same original gray material and the other half is darker. Although the two halves are coplanar, there is a clear value discontinuity in illumination across the edge separating the two materials. Now, consider a small opaque cube on the floor of the room and the shadow that it casts on the floor and possibly on the walls. The direct light from a point light source near the ceiling casts a hard shadow[4] of the cube onto the floor, and the transition from lit to unlit regions represents another illumination discontinuity in value. However, note that the light reflected off the ceiling acts as an area light source and casts a soft shadow[5] of the cube onto the floor. The transitions from lit to penumbra and from penumbra to umbra regions are illumination discontinuities in first derivative. Higher-order discontinuities are also possible [Heckbert92] but are not important for this work, because of their low visual impact on the final images.

The above description illustrates the need to represent the continuity and discontinuities of the radiosity function accurately in its domain. In most regions of the domain we may need smooth radiosity transitions between adjacent patches, whereas in other regions we may need to represent explicit discontinuities. Because it is hard to find a single reconstruction function for the entire scene combining all these features, the problem is broken down into small parts. Driven by the surface subdivision from the previous section, each final mesh element is associated with a continuous reconstruction or interpolation patch—a patch of an interpolation polynomial. In fact, each element is associated with three independent color patches; one for each color band (R, G, and B). However,

[4] A *hard shadow* contains hard edges—lit to unlit transitions.

[5] A *soft shadow* is composed of umbra and penumbra regions. The *umbra* is the completely unlit region, whereas the *penumbra* is the partially lit region.

as the treatment is analogous for each color component, radiosity is analyzed as a one-dimensional quantity—a per-wavelength quantity.

The simple association of a continuous reconstruction patch with each mesh element provides continuity inside independent patches/elements, but it does not ensure proper continuity between adjacent elements and across the entire domain. The continuity between adjacent patches is controlled by the order of the reconstruction scheme and by the sharing of information across and along edges of the adjacent patches. Given the assumption that a reconstruction patch is continuous inside its corresponding mesh element domain, note that the representation of radiosity discontinuities is only possible at element edges of the mesh.

The final result of the sampling phase in the previous section is usually a mesh of quadrilaterals and/or triangles representing the samples of the radiosity function. As previously assumed, each of these polygons represents the uniform radiosity in the corresponding area of the scene. Consequently, each polygon in the resulting mesh contains a single RGB triplet, which defines its color for rendering purposes. Therefore, a single flat reconstruction patch can be associated with each mesh element. This corresponds to a zero-order reconstruction scheme. Clearly, this zero-order polynomial scheme cannot guarantee any type of functional continuity along and across edges of adjacent polygons (except when two adjacent patches share the exact same RGB values), which causes a faceted appearance on the rendering of such scenes.

To increase functional continuity along and across shared edges of adjacent polygons, higher-order reconstruction schemes need to be used. After zero-order reconstruction, the next order up is bilinear interpolation—usually the standard for rendering radiositized scenes. Instead of using a single color value for each entire polygon, radiosity approximations are computed at the vertices of the polygons and then bilinearly interpolated inside the polygon. The radiosity at a vertex is given by the radiosity average of all the uniform elements sharing that vertex. Although bilinear interpolation can provide smoother results than the faceted shading, the results are still discontinuous in the first derivative of the reconstructed radiosity function.

The discontinuities in intensity and/or first derivative of the zero-order and of the bilinearly reconstructed functions in regions where the radiosity function should be smooth may appear as noticeable Mach banding artifacts[6] in the final images. Our visual system is sensitive to

[6] The perceptual effect known as *Mach banding* was first reported in 1865 by the Austrian physicist Ernst Mach [Ratliff72, Foley90]. The effect depends directly on the distribution of illumination and exaggerates the intensity change at any edge where there is a discontinuity in magnitude or slope of the illumination.

those discontinuities, and higher-order reconstruction schemes must be used to provide appropriate continuity on the reconstructed radiosity function between adjacent polygons. Image synthesis aims at C^1 continuous shading functions— continuous first derivatives—for representing smooth regions while avoiding Mach banding artifacts.

In terms of our radiosity reconstruction analysis, the next order up for functional reconstruction is biquadratic interpolation (Section 2.6.2.1). Although quadratic interpolation can provide first derivative continuity between adjacent elements it may produce fast transitions and overshooting in the interpolated functions (see Figure 5.5). In addition, quadratic interpolation is not able to capture inflection points between samples.

Figure 5.5: *One-dimensional quadratic (left) and cubic (right) interpolations compared to an arbitrary function (dashed): notice the fast transitions, the overshooting, and the inability to capture inflection points between samples of the quadratic interpolation compared to the original function. In contrast, notice the more controllable approximation obtained with cubic interpolation.*

The next interpolation order up (bicubic) provides higher control of the reconstructed function producing smoother transitions. According to Bastos et. al. [Bastos93] bicubic interpolation can provide the C^1 continuity needed for smooth radiosity rendering, i.e., the first derivative continuity of the reconstructed radiosity function across multiple adjacent patches.

All the reconstruction/interpolation schemes discussed above were based on the mesh of polygons resulting from the sampling phase of the previous section. Although these mesh-based rendering schemes are compatible with current graphics hardware, they easily overwhelm the rendering capabilities of most architectures, due to the huge amount of data produced by the adaptive subdivision of the scene. In addition, the current graphics hardware supports only zero- and first-order per-vertex interpolation of the attributes associated with mesh elements/polygons, which does not allow us to handle the bicubic interpolation of radiosity necessary for smooth renderings. The next section describes a more effective form of storing and rendering radiosity data without losing any information, a method which is also amenable to implementation with current graphics hardware and allows for higher-order interpolation/reconstruction schemes than does the hardware-assisted mesh-based scheme.

5.5 Radiosity As Textures—*RAT*

The elements-based radiosity solution and storage method described in the previous sections induced a radiosity rendering technique based on dense meshes of polygons with corresponding reconstruction or interpolation patches. This mesh-based rendering technique is effective because it enables the graphics hardware to perform the radiosity interpolation when rendering the corresponding polygons. However, a more careful analysis of the technique shows unnecessary use of geometrical entities to represent color (radiosity) information.

Suppose, for example, a triangular room floor represented with a single triangle that was recursively subdivided in a quadtree for radiosity representation. Not unrealistically, assume eight levels of subdivision and a resulting full quadtree[7]. A full quadtree with eight levels of subdivision generates $4^8 = 65536$ triangles. Clearly, in terms of geometry, there are 65535 unnecessary triangles in the final triangulated representation; note that all the 65536 are coplanar and together define the same triangular region represented by the original unsubdivided triangle. Of course, the additional triangles were created to locate radiosity samples inside the original surface, rather than to describe the geometry of the surface. Think of the unnecessary amount of geometry data that needs to cross the graphics pipeline for this mesh-based radiosity rendering approach.

Instead of sending a triangle for each radiosity sample in the subdivided mesh, geometry can be decoupled from illumination information and both sent separately to the graphics pipeline. In our example, for geometry it is desirable to send only the three original vertices describing the unsubdivided triangle, and for illumination we want to send the radiosity data resultant from the adaptive radiosity. Additionally, the radiosity data must be organized in some form that can be appropriately attached to the triangle. This suggests the use of texture mapping[8]. Essentially, one wants to calculate the distribution of light across the triangle and then map that as an image onto the triangle. The next sections describe techniques for converting radiosity data from a quadtree into a texture map, and reconstruction techniques for smoothly but still efficiently displaying the radiosity data [Bastos93, Bastos96, Bastos97].

[7] A full quadtree has all its leaf nodes at the deepest level of the tree.

[8] A *texture map* is an image-based primitive used to add detail to surfaces. The discrete elements of the texture map are called *texels* and represent the quantity to be mapped onto the primitive. A texture map is parameterized in texture space and maps to corresponding parameters on parameterized primitives.

5.5.1 From Full Quadtrees to Textures

The fairly regular sampling produced by quadtree adaptive subdivision can be effectively represented as a texture map, instead of a mesh of polygonal elements. For each quadtree representing adaptive sampling of the radiosity function across a surface, our technique creates a texture map representing the same information. Both the original quadtree radiosity and the produced texture map are floating point entities. Remember, also, that each node of a quadtree represents the uniform radiosity across a polygonal element of the mesh.

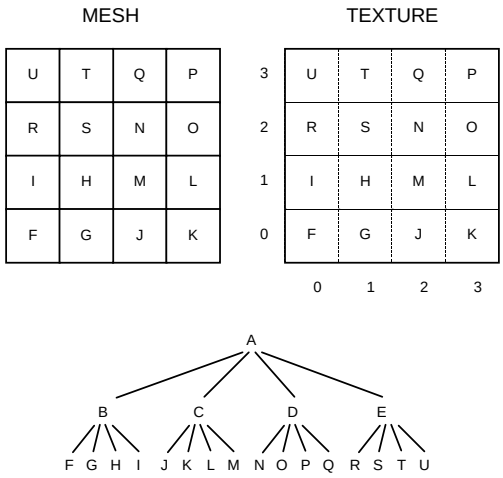


Figure 5.6: *Uniform subdivision of a quadrilateral, the corresponding full quadtree, and the mapping to a corresponding texture.*

On a one-to-one mapping, each texel in the texture corresponds to a unique leaf node in a quadtree (see Figure 5.6). The level of the deepest leaf node of a quadtree (the *height* of the quadtree) determines the resolution of the texture map. The number of texels in either side of a texture corresponding to a quadtree is given by

$$nu = nv = 2^{height}. \tag{5.6}$$

Note, then, that a square texture map with appropriate resolution is equivalent to a full quadtree, in terms of the number of texels and the number of leaf nodes.

A full quadtree is flattened into a texture map by mapping each of the quadtree leaf nodes into the corresponding texels in the texture map. The algorithm follows a depth-first traversal of the quadtree where actions are taken only at leaf nodes. The recursive algorithm for converting the

```
FullQuadtreeToTextureZeroOrder( quadtreeNode, level, height,  u, v )
{
   if ( level == height ) {   //  It's a leaf node
      texel[u][v] = quadtreeNode->radiosity
   }
   else {   //  It's an internal node, recurse.
      FullQuadtreeToTextureZeroOrder( quadtreeNode->child[0], level+1, height, u*2,   v*2)
      FullQuadtreeToTextureZeroOrder( quadtreeNode->child[1], level+1, height, u*2+1, v*2)
      FullQuadtreeToTextureZeroOrder( quadtreeNode->child[2], level+1, height, u*2+1, v*2+1)
      FullQuadtreeToTextureZeroOrder( quadtreeNode->child[3], level+1, height, u*2,   v*2+1)
   }
}
```

5.7: *Converting a full quadtree to a texture with a zero-order approach.*

radiosity quadtree of a quadrilateral surface into a texture is presented in Program 5.7, which is called for the root node of each quadtree with the following function call:

```
FullQuadtreeToTextureZeroOrder( quadtreeRoot, 0, quadtreeHeight, 0, 0 ).
```

Note that the `u,v` parameters indexing the texture are recursively shifted during the quadtree traversal for properly mapping leaf nodes to the appropriate texels. The algorithm for converting a quadtree associated with a triangular surface into a texture looks similar to the one above, except for the mapping recursively applied to the `u,v` parameters during the traversal.

The algorithm presented above applies for converting full quadtrees into textures in the zero-order scheme of per-polygon radiosity. That is, each leaf node of the full quadtree maps to a single texel in the texture map. The next sections discuss a technique for first-order approximation of the reconstructed radiosity function and for handling non-full quadtrees—both are desirable features for efficient rendering of adaptive subdivision radiosity.

5.5.1.1 A First-Order Approach

Instead of using a single radiosity value per polygon in a zero-order approximation of the radiosity function, radiosity methods, usually, compute radiosity at the vertices of the mesh to provide a first-order approximation of the radiosity function. Each per-vertex radiosity is approximated by taking the average of the radiosity of all the polygons sharing that vertex. The per-vertex radiosities are then bilinearly interpolated to provide the first-order approximation of the radiosity function. The same averaging and first-order radiosity approximation is possible using radiosity as textures without losing

any information and using a simpler approach, as the radiosity texture implies the necessary adjacency information not directly captured by meshes (follow in Figure 5.8).

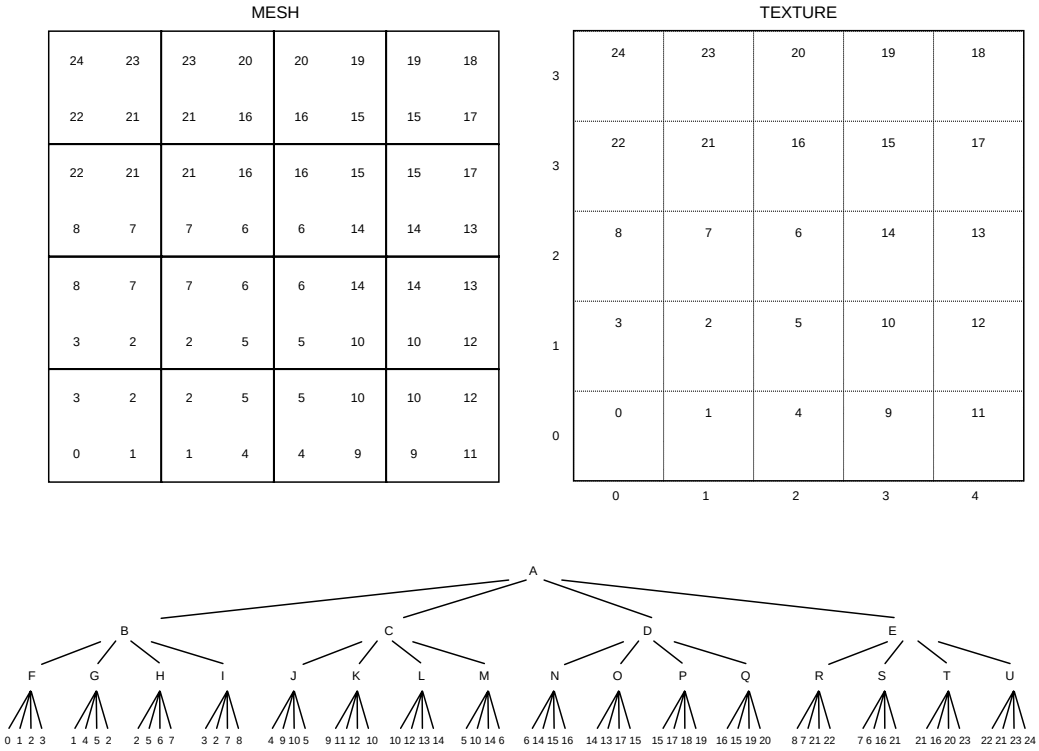


Figure 5.8: *Uniform subdivision of a quadrilateral with the corresponding shared vertices, the corresponding full quadtree, and the mapping to a texture in which each texel represents a vertex location in the mesh.*

In terms of the quadtree, each leaf node in the first-order scheme also now contains a set of vertices (four vertices for a quadrilateral and three for a triangle) for representing the respective geometry. Each of these vertices maps to a texel in the radiosity texture, which we refer to as a *vertex location* in the texture map. Note that now texels represent per-vertex radiosity, instead of per-element radiosity. Note also that a mesh of $n \times n$ quadrilaterals contains $(n+1) \times (n+1)$ vertices. Consequently, the number of texels on the side of a radiosity texture for a first-order approximation is given by Equation (5.6) incremented by one, so that the texture captures the total number of vertices instead of the number of elements in the mesh. Note also that adjacent leaf nodes of the quadtree have vertices that share/map to the same texel in the maps.

Starting from a zero-order (constant) representation of radiosity in a quadtree, the conversion process from a quadtree to a texture for a first-order scheme creates two two-dimensional maps, instead of a single one as in the zero-order scheme discussed in the previous section (follow in Figure 5.9).

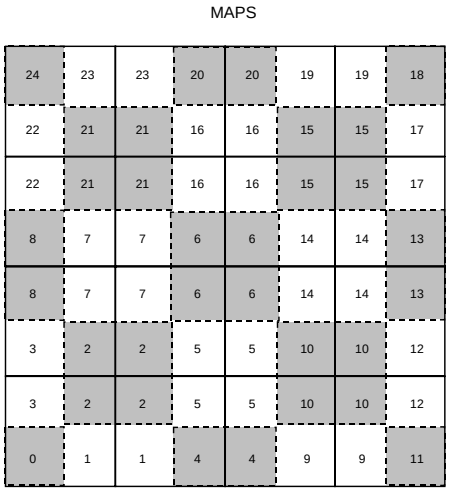


Figure 5.9: *Representation used for the two maps for computing average radiosity at vertex locations in Figure* 5.7*. The checkerboard pattern represents the texel regions of the maps organized on top of the mesh, and shows which elements are combined to compute the average radiosity at the corresponding vertex locations. The accumulator map contains at each texel the summation of the radiosities of the mesh elements sharing that texel of the maps. The counter map contains at each texel the number of mesh elements sharing that texel in the map. For example, the texels at the four corners of the maps have only one element; texels along the edges have two elements; and texels inside the maps have four elements.*

The need for two maps comes from the average radiosity that needs to be computed. Shared vertices of adjacent leaf nodes map to the same texels/vertex locations in the map. Both maps have the same resolution and serve to compute the average radiosities. One of the maps, *the accumulator*, is used for accumulating radiosity at the texels touched by leaf nodes sharing/mapping to that same texel. The other map, *the counter*, counts the number of leaf nodes sharing each texel (contributing to the accumulated radiosity at each texel). During a depth-first traversal of the quadtree, each leaf node contributes to up to four texels in the maps corresponding to a quadrangular surface (six texels for a triangular surface). The exact texels to be updated by each leaf node are determined by the recursive mapping functions as described above. For each leaf node, the corresponding texels in the accumulator map have their radiosity augmented by the node's radiosity and the corresponding texels in the counter map have their values incremented by one. Using these two maps, it is then possible to average the constant-basis radiosity from leaf nodes to approximate the radiosity at vertex locations: both maps are sequentially traversed, and, whenever the counter texel is not zero, the accumulator texel is divided by the counter.

The recursive algorithm for converting the radiosity full quadtree of a quadrilateral surface into a texture for a first-order approximation is very similar to the zero-order approximation algorithm from

page 80. The recursive mechanism remains the same, but now each leaf node maps to four texels in the map (one for each vertex) instead of a single one (Program 5.10). Program 5.10 is invoked for the

```
FullQuadtreeToTextureFirstOrder( quadtreeNode, level, height,  u, v )
{
   if ( level == height ) {   //  It's a leaf node
      accumulator[u][v] += quadtreeNode->radiosity
      accumulator[u+1][v] += quadtreeNode->radiosity
      accumulator[u+1][v+1] += quadtreeNode->radiosity
      accumulator[u][v+1] += quadtreeNode->radiosity
      counter[u][v] += 1
      counter[u+1][v] += 1
      counter[u+1][v+1] += 1
      counter[u][v+1] += 1
   }
   else {   //  It's an internal node, recurse.
      FullQuadtreeToTextureFirstOrder( quadtreeNode->child[0], level+1, height, u*2,   v*2)
      FullQuadtreeToTextureFirstOrder( quadtreeNode->child[1], level+1, height, u*2+1, v*2)
      FullQuadtreeToTextureFirstOrder( quadtreeNode->child[2], level+1, height, u*2+1, v*2+1)
      FullQuadtreeToTextureFirstOrder( quadtreeNode->child[3], level+1, height, u*2,   v*2+1)
   }
}
```

Program 5.10: *Converting a full quadtree to a texture with a first-order approach.*

root node of each quadtree with the following function call:

```
FullQuadtreeToTextureFirstOrder( quadtreeRoot, 0, quadtreeHeight, 0, 0 ).
```

In addition, after the two maps (accumulator and counter) were computed, the radiosity average at vertex locations is trivially computed with the following algorithm:

```
for (u=0, u<nu, u++)
   for (v=0, v<nv, v++)
      if (counter[u][v] > 1)
         avrg_radiosity[u][v] = accumulator[u][v] / counter[u][v]
      else
         avrg_radiosity[u][v] = accumulator[u][v]
```

Program 5.11: *Computing radiosity average at vertex locations from the accumulator and counter maps.*

All the previous descriptions have assumed full quadtrees. However, the quadtrees resulting from adaptive subdivision are in general non-full quadtrees. In that case, a leaf node not at the deepest

level of the quadtree maps to more than one texel—maps to a region—in the texture map (Figure 5.13). Non-full quadtrees, as will be discussed in the next section, require an additional step to fill in uncovered regions of the texture map (or leaf nodes that map to more than one texel).

5.5.2 From Non-Full Quadtrees to Textures

The algorithms in the previous section were designed for full quadtrees. For non-full or general quadtrees, the recursive mechanism remains the same, but there are two things that need to be changed in the algorithm. First, the mapping function needs to be adjusted for leaf nodes at different levels of the tree. The full quadtree approach relied on the fact that all leaf nodes were at the deepest level of the tree; each leaf node was mapped into its corresponding texel by always starting at the texture origin and recursively shifting location until reaching the node's final position. Since for a full quadtree all leaf nodes are at the same level, the mapping function depended only on the height of the tree (captured by the number of recursive calls performed by the algorithm for each leaf node). For non-full quadtrees the mapping function depends both on the height of the tree and on the level of the leaf node on the tree. Clearly, leaf nodes at the deepest level of the tree should map to the same location as in the full quadtree approach. Leaf nodes at internal levels should map to a texel corresponding to one of the leaf nodes' children in a full quadtree, i.e., should not map to any texel of the leaf nodes' siblings or of their children (follow in Figure 5.13).

The recursive algorithm for converting the radiosity quadtree of a quadrilateral surface into a texture for a zero-order approximation is similar to the zero-order approximation algorithm for full quadtrees in page 80. The recursive mechanism remains the same, but now the mapping function needs to take into account the relative level of each leaf node with respect to the height of the quadtree (Program 5.12).

Secondly, if a full quadtree has as many leaf nodes as texels in a texture of given resolution, a corresponding non-full quadtree contains fewer leaf nodes than texels in the same texture. Obviously, mapping each leaf node of the non-full quadtree to a single texel in the texture leaves some texels untouched. These gaps in the texture map can be filled in by interpolation. A zero-order approximation finds the coverage of a leaf node in the texture map and replicates the radiosity value of that leaf node into all those texels. A first-order approximation uses the radiosity at the corners (four or three, depending on the shape of the original node) of the leaf node, to bilinearly interpolate values inside the hole. The recursive algorithm for filling in the gaps on the radiosity texture is presented in

```
QuadtreeToTextureZeroOrder( quadtreeNode, level, height,  u, v )
{
   if ( level == height ) {   //  It's a leaf node
      span = 2^(height - level)
      u' = u * span
      v' = v * span
      texel[u'][v'] = quadtreeNode->radiosity
   }
   else {   //  It's an internal node, recurse.
      QuadtreeToTextureZeroOrder( quadtreeNode->child[0], level+1, height, u*2,   v*2)
      QuadtreeToTextureZeroOrder( quadtreeNode->child[1], level+1, height, u*2+1, v*2)
      QuadtreeToTextureZeroOrder( quadtreeNode->child[2], level+1, height, u*2+1, v*2+1)
      QuadtreeToTextureZeroOrder( quadtreeNode->child[3], level+1, height, u*2,   v*2+1)
   }
}
```

Program 5.12: *Converting a quadtree to a texture with a zero-order approach.*

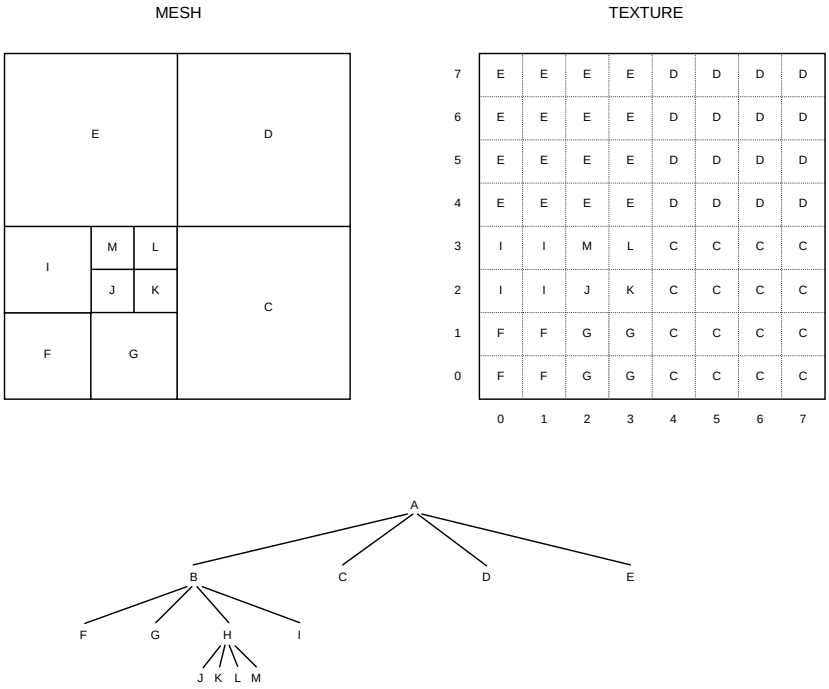


Figure 5.13: *Adaptive subdivision of a quadrilateral, the corresponding non-full quadtree, and the mapping to a corresponding texture. Notice the empty texels in the texture not covered by any leaf node using our algorithm.*

Program 5.14. Note that this algorithm could be consolidated with the algorithm on page 80 into a

```
FillInGapsZeroOrder( quadtreeNode, level, height,  u, v )
{
   if ( quadtreeNode->numberChildren == 0 &&
        level != height ) {   //  It's an internal leaf node
      span = 2^(height - level)
      v' = v * span
      for (i=0; i<span; i++, v'++ ) {
         u' = u * span
         for ( j=0; j<span; j++, u'++ ) {
            texel[u'][v'] = quadtreeNode->radiosity
         }
      }
   }
   else {   //  It's an internal node, recurse.
      FillInGapsZeroOrder( quadtreeNode->child[0], level+1, height, u*2,   v*2)
      FillInGapsZeroOrder( quadtreeNode->child[1], level+1, height, u*2+1, v*2)
      FillInGapsZeroOrder( quadtreeNode->child[2], level+1, height, u*2+1, v*2+1)
      FillInGapsZeroOrder( quadtreeNode->child[3], level+1, height, u*2,   v*2+1)
   }
}
```

Program 5.14: *Filling in the gaps with a zero-order approach.*

single algorithm to accomplish both tasks. For clarity, we keep them as separate operations.

This section analyzed the zero-order algorithms for mapping leaf nodes of a non-full quadtree into a texture and for filling in the occasional gaps in the resulting texture map. First-order algorithms are also possible and follow the same modifications as presented in Section 5.5.1.1 imposed on the algorithms of this section.

5.5.3 Resampling and Filtering

The rendering of radiosity data is a resampling process (Section 2.6). It starts with sampled radiosity data on the surfaces of a scene and generates pixels on the screen. As such, radiosity rendering involves both minification[9] and magnification[10] processes. Since that radiosity data usually contains mostly low

[9] *Minification* happens whenever a sampled signal is mapped into a lower resolution space—information is lost.

[10] *Magnification* happens whenever a sampled signal is mapped into a higher resolution space—information needs to be created/reconstructed.

spatial frequencies, this dissertation only analyzes the magnification process. [Wolberg92] gives an analysis of filtering techniques for minification.

Magnification when rendering radiosity data happens both for the mesh-based approach and for the texture-based approach; the spatial resolution on the screen is usually higher than the sampling resolution on the radiosity domain. The graphics hardware performs either zero-order (nearest neighbor) or first-order (bilinear) interpolation when magnifying radiosity data. These low-order interpolation schemes cause value and slope discontinuities in the reconstructed signal (Section 5.4). Perceptually, these undesired discontinuities produce faceting and Mach banding in the rendered images. The naive solution for this problem is to oversample the radiosity domain such that radiosity discontinuities and perceptual effects are minimized. However, this solution hurts performance by creating unnecessary data.

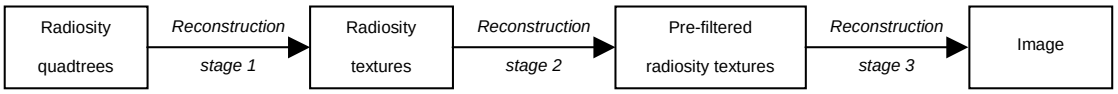


Figure 5.15: *Three-stage radiosity reconstruction pipeline.*

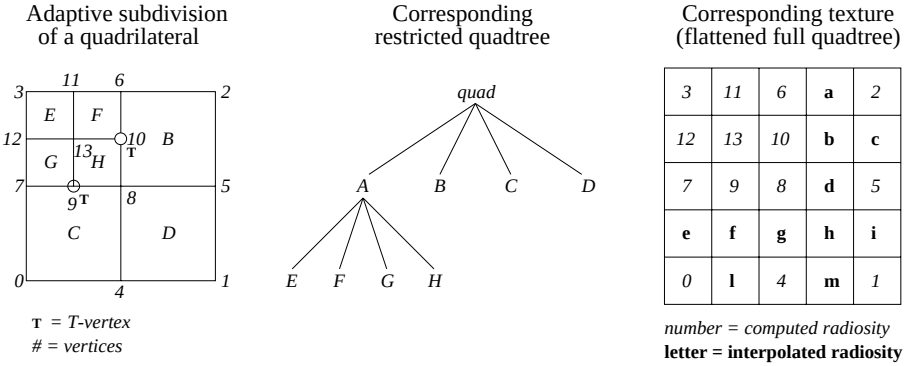
The next three sections present a three-stage reconstruction pipeline for radiosity data that reduces or completely eliminates the perceptual artifacts by using successive reconstruction and filtering of the data until they are suitable for display (Figure 5.15). The pipeline starts by resampling given radiosity data; each radiosity quadtree (balanced or unbalanced) is converted into a single continuous reconstructed radiosity function which is then resampled to the original sampling rate. This first reconstruction stage handles T-vertices[11] and unrestricted quadtrees[12] in a unified way. Then, in reconstruction stage 2, the resampled radiosity data are low-pass filtered. This second stage prepares the radiosity data for rendering. The last reconstruction stage is the interpolation/filtering scheme used for rendering the radiosity data. Since the radiosity data were already low-pass pre-filtered in the previous stages, the interpolation/filtering order for the third stage can be as low as first-order (standard bilinear shading) and still produce results with minimal artifacts. These approaches are discussed in terms of the radiosity-as-textures technique presented in Section 5.5, but also apply to the mesh-based technique (even though with much more elaborate implementations).

[11] A *T-vertex* refers to a vertex not shared by all the polygons sharing its supporting edges.

[12] In a *restricted quadtree*, adjacent leaf nodes are never more than one tree level apart.

5.5.3.1 Reconstruction Stage 1—Resampling Radiosity from Any Quadtree

The first reconstruction stage is a unified solution for the T-vertex and unrestricted quadtree problems. Radiosity subdivision can generate unbalanced quadtrees with nonconforming[13] elements in regions with different mesh densities [Cohen93, Sillion94]. Figure 5.16 presents an example where neighboring elements have different levels of subdivision, creating T-vertices in the mesh (e.g., vertices 9 and 10 in the figure). T-vertices are undesirable because the interpolated values at those points



3	11	6	**a**	2
12	13	10	**b**	**c**
7	9	8	**d**	5
e	**f**	**g**	**h**	**i**
0	**l**	4	**m**	1

Figure 5.16: *Adaptive subdivision of a quadrilateral, the restricted quadtree, and the corresponding texture map: notice the T-vertices number 9 and 10.*

(interpolation between vertices 6 and 8 for element B at location of vertex 10) will potentially be different from the radiosity value computed at the same point (at vertex 10 for elements F and H). This generates a visual discontinuity along the edges supporting that vertex (edges 6–10 and 10–8).

Several solutions to the T-vertex problem have been proposed in the finite-element literature and the most important are [Cohen93]: replace the computed value with the interpolated value at that point; or use restricted quadtrees and triangulate the larger element that neighbors the T-vertex. Our solution is based on a recursive bicubic patch-based reconstruction. The technique does not subdivide the original elements, keeps all the original radiosity samples, and creates a continuous radiosity representation over the original polygon (root of the quadtree).

We begin by creating a bicubic patch per color component for every leaf in the quadtree. This provides C^1 continuity between adjacent patches at the same quadtree level, but does not ensure C^1 continuity along edges containing T-vertices. Then, bicubic patches at leaf nodes that are not at the deepest level of the quadtree are recursively midpoint subdivided until we get a full quadtree. In that subdivision process, previously computed radiosity values are shared by the new patches, and radiosity

[13] Two adjacent elements in a quadtree are nonconforming if they are at different levels of the tree; this is the cause of T-vertices.

values for new points are approximated by interpolating the bicubic patch at the level directly above in the quadtree. This process is recursively applied until we have a full quadtree that can be flattened into a texture (Figure 5.16–right). This technique generalizes directly for unrestricted or unbalanced quadtrees, handling T-vertices in the same simple and hierarchical way. Note that this algorithm implies the creation of a full quadtree from any non-full quadtree. Note also that this algorithm is based on the interpolation-patches reconstruction scheme presented in Section 5.4 and the mechanism of algorithm 5.12 from page 85.

Given the continuous radiosity reconstructed function, it is now possible to resample the original data at any appropriate spatial resolution and to provide smooth radiosity renderings. Current graphics hardware, however, implements only zero- and first-order interpolation schemes—not bicubic—which cannot provide the smoothness required for radiosity rendering. However, it is possible to resample the continuous data and generate radiosity textures as described in previous sections, which are then taken into the next reconstruction stage and ultimately to smooth rendering with the first-order interpolation hardware.

5.5.3.2 Reconstruction Stage 2—Low-Pass Pre-Filtering Sampled Radiosity

The previous stage created radiosity textures sampled from continuous reconstructed functions. Given the continuity of the reconstructed function, the radiosity texture can be up-sampled and provide smoother data than the original radiosity data. However, this approach would be similar to oversampling the radiosity domain, and just as undesirable for performance reasons.

Alternatively, we want to produce processed radiosity data at the same given resolution but still provide smoother results. This involves recognizing that some of the radiosity rendering artifacts are due to undesired high spatial frequencies from low-order interpolation schemes. That is, the traditional zero- and first-order interpolation shading introduces high frequencies in the spectrum of an image due to the value and slope discontinuities of the low-order interpolation. A solution is to low-pass filter the radiosity data provided by the previous section such that high spatial frequency content created in the shading process is minimized. The blur produced by low-pass filtering the radiosity data reduces or completely eliminates the discontinuity artifacts of the renderings, even when bilinear shading is used for rendering the image in stage 3 of the next section.

We have experimented with low-pass filters ranging from a 2×2 bilinear filter up to 5×5, 4^{th} order Gaussian filter. Increasing the resolution of the filter increases the smoothness of the results

with, consequently, an increased blur of the whole texture. Increasing the resolution of the filter also increases the computational time to convolve the filter with the textures.

5.5.3.3 Reconstruction Stage 3—Radiosity Shading

The previous stage preprocessed the radiosity data for this final stage of reconstruction in our pipeline—the final radiosity rendering. We again start with sampled data in the form of a texture map and reconstruct intermediate texel values on the screen as we render the textured polygon.

The usual reconstruction technique for rendering textures is bilinear interpolation. This type of interpolation is readily available and fast on current graphics hardware. However, bilinear interpolation cannot represent first derivative continuity across the entire texture, which inability may cause artifacts on the final images (Section 5.4). We have explored two alternatives here. First we tried taking the results from the first stage reconstruction of Section 5.5.3.1 directly into the radiosity shading and using bicubic texture filtering available on *SGI*^{TM} hardware [Bastos97]. In that case, we replaced the default bilinear texture filtering by a bicubic filter (1/3, 1/3) [Mitchell88]. Then, we expanded this two-stage reconstruction pipeline into a three-stage one, as described here. In the three-stage pipeline radiosity data resulting from stage 1 is pre-filtered before getting to the final radiosity shading stage. This three-stage approach produced renderings as smooth as the two-stage one, without the performance penalty of the on-the-fly bicubic filtering. The advantages of using the bilinear filtering over the bicubic filtering are higher performance, and reduced frame buffer and texture memory requirements; standard bilinear interpolation operates with 8 bits per color component, whereas the bicubic filtering requires more bits per component to avoid possible range overshooting of the bicubic interpolation. The graphics API only offered 12 bits/component as an alternative to 8 bits/component.

5.5.4 Curved Objects

We have heretofore assumed planar (triangular or quadrilateral) primitives. Computing and displaying smooth radiosity for curved primitives poses a more interesting problem. This section extends the radiosity-as-textures techniques described in the previous sections to handle radiosity data for curved surfaces efficiently.

The applicability of our approach to u,v-parameterized curved surfaces, such as Bezier or NURBS patches, is straightforward, and it works exactly in the same way as for quadrilaterals or triangles. The curved patch is partitioned for radiosity computation, and the corresponding quadtree

is used for deriving the corresponding RAT to be applied as a texture to the u,v-parameterized curved surface. The approach also applies in the same way to implicit surfaces.

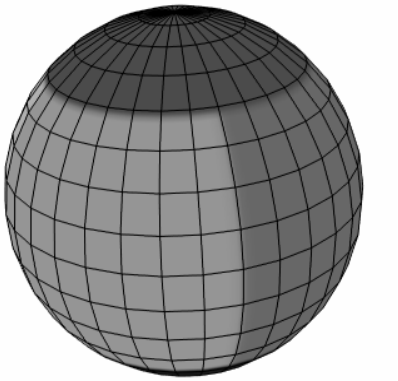


Figure 5.17: *Sphere subdivided into six patches created by the projections of the faces of a cube inscribing the sphere.*

We have applied our radiosity-as-textures algorithms to a sphere subdivided into six equal-area spherical patches, as shown in Figure 5.17. Each of the patches is independently handled during radiosity computation as a quadrilateral patch. This means that each patch has a corresponding quadtree subdivision after radiosity computation. The challenge here is to reconstruct a smooth radiosity function over the sphere by sharing information along and across the edges between patches. The RAT reconstruction scheme for the six-patch sphere works as follows:

> Create a radiosity texture for each spherical patch, independently, by traversing their quadrilateral-shaped quadtrees as described in sections 5.5.1 and 5.5.2. Then, create a final radiosity texture by combining and resampling each of the independent patch radiosity textures. The mapping and distribution of radiosity to be stored in the final radiosity texture depends on the type of texture coordinates to be used by the rendering engine. Our final radiosity texture for a sphere uses texture coordinates compatible with current graphics libraries such as *OpenGL*, which represents spherical coordinates in the latitude-longitude system. Latitude is represented by the vertical axis in texture coordinates and ranges from $-\pi/2$ to $+\pi/2$ from bottom to top in the texture. Then, each latitude value corresponds to a line in the texture. The horizontal dimension of the texture represents longitude. Note the varying spatial resolution in longitude for different fixed latitudes. For example, the poles of the sphere represent single points in the sphere; however, they have

the same spatial resolution as the equator, which is represented by the middle vertical line in the texture.

Let's arrange the independent radiosity textures on a plane as presented in Figure 5.18. This clearly needs some warping and resampling to create a texture compatible with OpenGL.

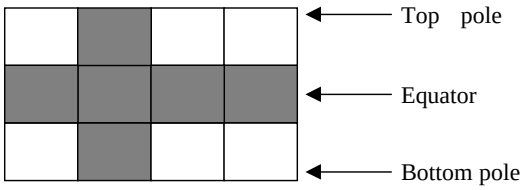


Figure 5.18: *Layout for the six radiosity textures from the sphere.*

Given the potentially different resolutions of the independent radiosity textures (quadtree) among the six patches, we start by determining the maximum resolution among them. This maximum resolution defines the resolution of the final radiosity texture such that it accommodates all the information captured on the independent radiosity textures without any down-sampling. We make the horizontal resolution four times the maximum resolution of the independent radiosity textures and the vertical resolution two times the same maximum resolution. The factor of four in the horizontal resolution comes from the need for the final radiosity texture to accommodate information from four independent radiosity textures along the equator, as suggested by figures 5.17 and 5.18. The factor of two for the vertical resolution is somewhat more complicated. Vertically, we need to accommodate the two poles and the equatorial region. Figure 5.18 suggests a factor of three in the vertical resolution. However, we should notice that the pole patches in that figure use only one quarter of the total horizontal available resolution. They need to be spread so that they use the whole top and bottom regions of the texture. Simply resampling the top/bottom pole so that it uses the whole top/bottom row wastes resolution. To compensate, we use only half the vertical resolution at polar regions. Even though this still represents more resolution than may be necessary at the poles, it facilitates the smoothing process of the final RAT on the pole regions and achieves compatibility of the radiosity textures with *OpenGL* texture requirements (power of two resolutions on the texture sides).

Once the resolution for the final radiosity texture has been determined, we resample the independent radiosity textures to create the final texture. We backward map all the texels from the final radiosity texture into the corresponding independent textures and point-sample the corresponding RAT to get the radiosity value at that location.

After the independent radiosity textures are resampled to generate the final radiosity texture, the final texture is low-pass filtered as described in Section 5.5.3.2. As continuity is needed along and across the texture seams on the sphere, padding of the final radiosity texture is necessary so that texels from the left side of the texture are taken into account when filtering its right side, and vice-versa. The padding for the polar regions is done by replicating the first and the last rows of the textures. The size of the padding region (number of texels) follows the requirements of OpenGL convolution [Woo96].

The techniques developed here apply for other curved surfaces. The partitioning of the surface into patches is necessary for the radiosity solution process, and the reconstruction steps use that information to handle continuity across the surface. The mapping of the radiosity information into a single texture, the padding, and the pre-filtering are necessary to ensure smooth rendering of the radiosity texture across the entire curved surface.

5.5.5 Performance of RAT Models

We observed that replacing polygonal radiosity meshes with texture maps can dramatically improve rendering performance, even across a variety of real-world models that contain mixes of meshes and textures [Bastos96]. Table 5.5.5 presents results for the grotto model (Figure 5.19). The table lists the number of polygons for the original (`no radiosity`) model—before radiosity solution, for the adaptively subdivided (`adapt.subdiv.`) model—after radiosity solution, and for the *RAT* model rendered with bilinear and bicubic shading. The table also compares the size of the models and the use in texture memory for *RAT* models and presents the frame rates for each model.

Model	Polygons	Size	Textures		Frame
	number	(bytes)	memory	number	rate
`no radiosity`	452	72K	0 MB	0	60.0
`adapt.subdiv.`	93,640	16,282K	0 MB	0	3.2
`RAT bilinear`	452	93K	4 MB	358	25.3
`RAT bicubic`	452	93K	4 MB	358	13.3

Table 5.1: *Performance with the* `grotto` *model.*

5.5.6 Appearance of Textures Reconstructed with One- and Two-Stage Pipeline

5.5.6.1 Results for the two-stage reconstruction pipeline

Images (c)-(e) in Figure 5.20 present a set of images demonstrating the reconstruction improvements obtained with our two-stage pipeline. Notice how a first stage bicubic reconstruction can reduce

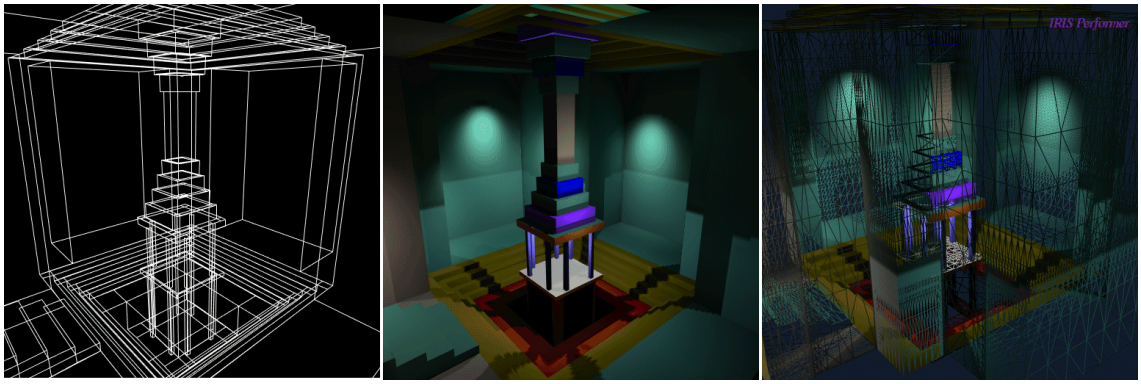


Figure 5.19: *The grotto model comparing the geometry before radiosity (on the left), the adaptively subdivided model after radiosity computation (on the right), and the final rendered result equivalent for both mesh and texture radiosity (in the middle). Note that the radiosity-as-textures version of the model renders the same number of polygons as the original unsubdivided model. Note also that the reduction in polygon count from the adaptively subdivided model to the radiosity-as-textures model translates into texture memory usage.*

the artifacts resulting from undersampled radiosity solutions. Image (a) is the radiosity texture corresponding to the quadtree (height 2, 5×5 samples) output from Lightscape^{TM}. Image (a) is used to reconstruct textures (c), (d), and (e) using bilinear (d) and bicubic interpolation ((c) and (e)). Image (d) presents a standard rendering of the radiosity information in image (a) generated by bilinearly resampling image (a) to 256×256 pixels. The image shows a prominent Mach band (star-shaped) artifact in the center of the bright spot in the upper left corner due to slope discontinuities in that region. Image (e) presents the resampling of image (a) with bicubic interpolation and how it can eliminate the artifacts of bilinear interpolation. Notice that the overall shape of the bright spot is rounder and there is no apparent star-shaped artifact in the center of the spot, as expected from the reference image (b).

Figure 5.20.(g) demonstrates the use of hardware bicubic interpolation of texture maps to produce a rendered image that is smoother and shows fewer shading artifacts than the corresponding bilinearly interpolated Figure 5.20.(f). Image (c) shows the resampling of the bicubic representation to 8×8 pixels (OpenGL allocates texture memory assuming powers of 2 pixels on the side of the textures–we scale up the 5×5 information to use all the allocated memory). Image (f) shows that even a texture map that has been generated from resampling C^1-continuous bicubic patches can show artifacts if the texel values are interpolated using bilinear mode in hardware. Notice that the bicubic filtering mode (image (g)) produces much smoother results by blurring the information. Notice also that bicubic filtering is three times slower than bilinear filtering on the machine described above, which

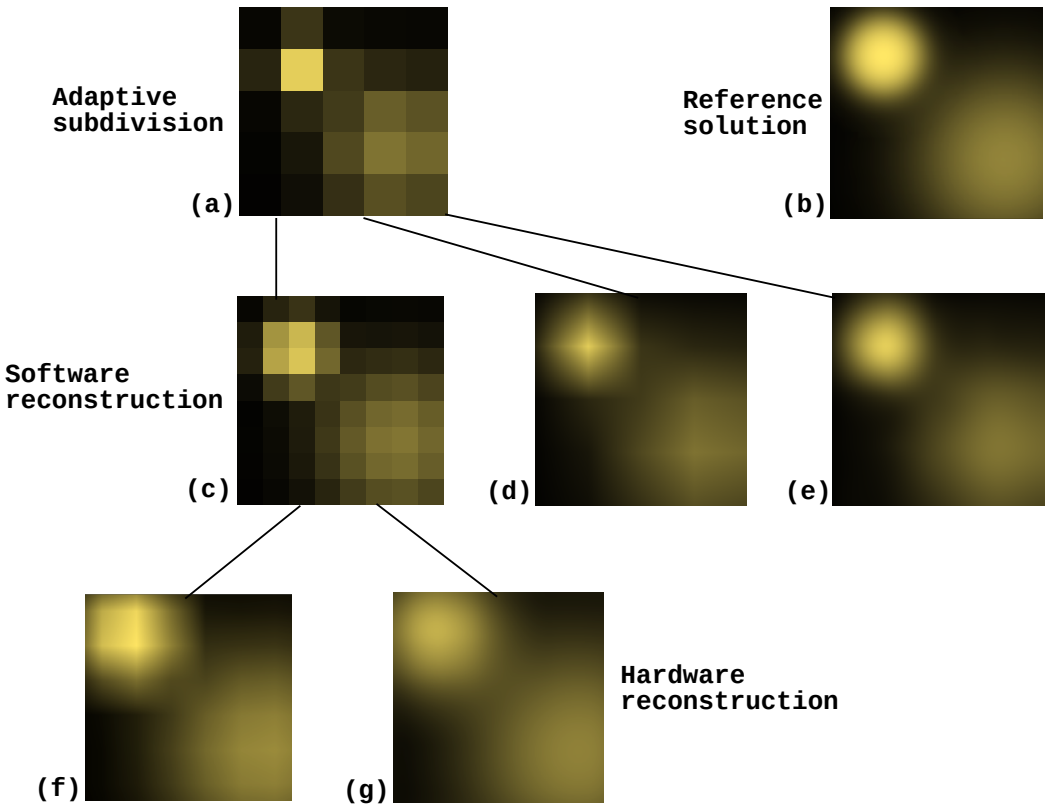


Figure 5.20: *Reconstruction results for the two-stage pipeline: (a) initial information available from two levels of subdivision in Lightscape*^{TM}*; (b) reference image obtained from a height 9 quadtree (up to* 513×513 *samples) to be used as a goal for the reconstructed images; (c) first stage of (software) bicubic reconstruction of image (a) with* 8×8 *resampling; (d) first stage of (software) bilinear reconstruction of image (a) with* 256×256 *resampling; (e) first stage of (software) bicubic reconstruction of image (a) with* 256×256 *resampling; (f) hardware bilinear filtering of image (c) with* 256×256 *resampling; and (g) bicubic hardware filtering of image (c) with* 256×256 *resampling.*

impacts performance. The next section presents results for our three-stage reconstruction pipeline, which provides smoother results than standard radiosity shading without impacting performance.

5.5.6.2 Results for the three-stage reconstruction pipeline

The major difference between the two-stage and the three-stage pipeline is an additional filtering stage in-between the two stages of the two-stage pipe. The additional stage prefilters the radiosity texture from the first stage to prepare it for shading (the last stage in both pipes).

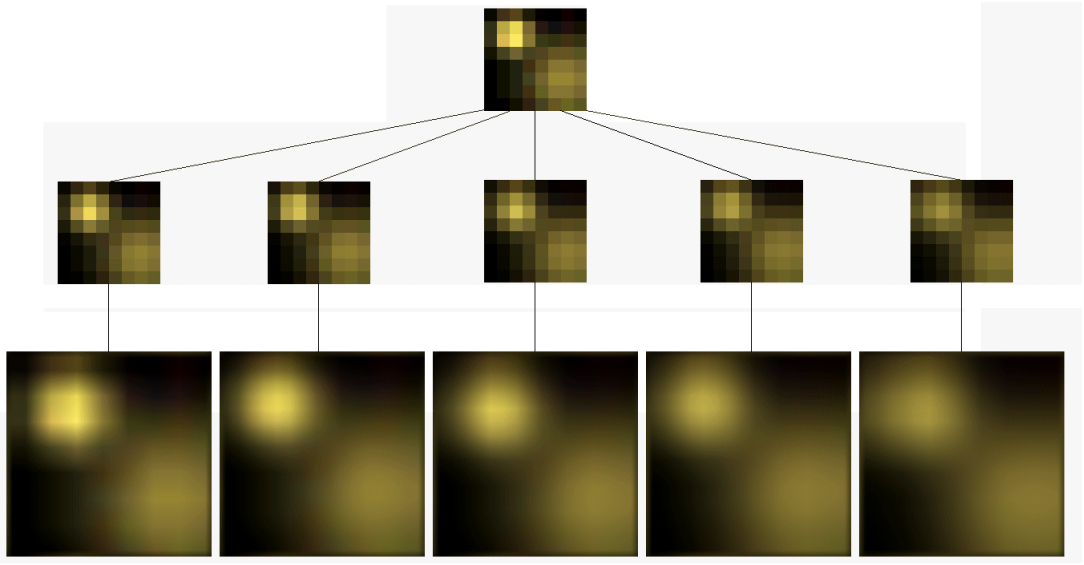


Figure 5.21: *Reconstruction results for the three-stage pipeline: the top most image represents the radiosity texture generated in the first stage by converting quadtree information into a texture; the second row of images shows the low pass filtering of the top image to produce prefiltered radiosity textures in the second stage of the pipe (filters range left to right from* 2×2 *to* 6×6 *samplings of a Gaussian filter); the third row of images shows the final result of the three-stage pipeline produced with standard bilinear interpolation of the prefiltered radiosity textures in the third stage of the pipe. The images in the first and the second row are all* 8×8 *in resolution and were scaled to* 128×128 *for display in the figure. The images in the third row are all* 256×256 *in resolution.*

Notice that the left most image in the second row of images corresponds to the standard radiosity rendering—bilinear interpolation with no prefiltering. Notice also the increasing blur for increasing filter resolution (from left to right in both the second and the third rows). The low-pass filtering performed by the sampled Gaussian filters reduces the high spatial frequency content of the radiosity texture before sending the texture to the standard bilinear shading stage. This low pass filtering implies smaller slope discontinuities in the bilinearly shaded textures, which translate into less noticeable artifacts in the final images. Figure 5.21 presents results for our three-stage radiosity reconstruction

pipeline. The top most image is the 8×8 radiosity texture output from the first stage (equivalent to image (c) in Figure 5.20). The second row of images shows the results of low pass filtering the top most image with a Gaussian filter with resolution ranging from 2×2 (bilinear interpolation) to 6×6 samples. The third row of images shows the results of rendering the prefiltered textures of the second row using standard bilinear shading in the graphics hardware. Compare the results in the third row with the reference image (b) in Figure 5.20.

Observe, however, that the prefiltering approach achieves smoother results by blurring the radiosity information (similar to what was achieved in image (g) of Figure 5.20). An excessively wide filter may result in an excessively blurred image, which may eliminate small features in radiosity textures when trying to smooth features of larger spatial size. Spatially-variant low pass filtering may be necessary for smoothing the textures but still preserving features of different sizes in a single radiosity texture. Note also that the three-stage pipe does not impact rendering performance, if compared to standard radiosity shading, but still produces smooth radiosity renderings.

5.6 Discussion

The radiosity-as-textures approach (as well as any other texturing technique) is a feed-backward mapping approach, whereas the mesh-based approach is a feed-forward mapping approach. The mesh-based radiosity approach forward-maps all the triangles onto the screen, whereas the texture-based approach forward maps a single triangle onto the screen and then looks up (backward maps) the corresponding radiosity value from the texture for each pixel covered by the unsubdivided triangle. If the unsubdivided triangle covers the same number or more of pixels on the screen as we have texels in the radiosity texture, the two approaches have similar costs. However, if the number of pixels covered by the unsubdivided triangle is smaller than the number of texels, the feed-backwards approach is more efficient than the feed-forward approach. For example, suppose that the unsubdivided triangle from our example in Section 5.5 covers 100 pixels on the screen. The feed-forward approach still needs to forward-map all the $65,536$ triangles onto the screen, whereas the feed-backwards approach forward-maps a single triangle and then performs only 100 backward maps to get the corresponding radiosity values. Clearly, the RAT approach can be more efficient than the mesh-based radiosity. As presented in this chapter, the RAT approach can also provide the same or better reconstruction schemes than the mesh-based approach.

Representing radiosity with texture is not a new idea. The next paragraphs briefly distinguish related approaches from our radiosity-as-textures approach.

Heckbert [Heckbert90] proposed the use of texture mapping as an alternative to the mesh-based radiosity approach. In his approach the energy arriving at a surface is computed using Monte Carlo ray tracing and stored at texels of radiosity textures (*rexes*). Uniform adaptive sampling is supported that organizes rexes into quadtrees. During rendering, radiosity at any pixel in the final image is approximated using bilinear interpolation. Due to the Monte Carlo ray tracing and the bilinear reconstruction step, Heckbert's approach can generate noisy and discontinuous images.

Myszkowski and Kunii [Myszkowski94] describe a method to replace the most complex radiosity mesh areas with texture maps based on available texture memory and the number of mesh elements that would be eliminated. Each selected surface is rendered as a Gouraud-shaded polygonal mesh, and a texture map is retrieved directly from the surrounding rectangular region in the frame buffer. In situations where there are mesh-based artifacts in the lighting solution, they recalculate the solution directly to the texture map itself, by applying radiosity sampling at locations corresponding to texels. Radiosity texture maps are rendered using bilinear texture interpolation on a Silicon Graphics RealityEngine.

Möller [Moller96] describes a method for replacing radiosity solutions for NURBS (geometry) models with a single texture map. This allows for multiple different levels-of-detail of the model with the same illumination texture map. Textures are generated from a radiosity mesh, and texels that have no corresponding mesh values are filled in using bilinear interpolation. The textured models are rendered on a Silicon Graphics RealityEngine using bilinear texture interpolation.

CHAPTER 6

NON-SCATTERED TRANSFERS OF LIGHT

The scattering of light spans a range of behaviors (Chapter 3). At one end, a single ray of light impinging on a surface is transferred with equal distribution in all outgoing directions—the ideally diffuse transfers. On the other extreme, light from a single ray at a given incoming direction is transferred into a single outgoing direction—the ideally specular transfers. Chapter 5 addressed the ideally diffuse transfers—the one end of the range. This chapter deals with the ideally specular transfers—the other end of the range. The in-between transfers—namely, the glossy transfers—are discussed in Chapter 7.

Chapter 5 considered the view-independent component of the light transport problem. Although important, that component lacks view-dependent specular effects, which considerably enhance the photorealism of a scene as it is viewed dynamically. These non-static shading and visual cues produce more photorealistic and more interesting results for interactive walkthroughs. This and the next chapter discuss view-dependent components of the light transport problem.

6.1 Ideally Specular Transfers

Ideal or non-scattered light transfers were discussed in Section 3.5. Both ideal reflections and ideal transmissions were considered. This chapter discusses algorithms to compute ideal reflections and leaves the ideal transmissions as an extension of the presented techniques.

The light transport Equation (4.1) is greatly simplified under the assumption that the light transfers are ideally specular. For ideally specular transfers, the BDF is a delta function in the directional space centered at the point in question—the BDF is non-zero only in the specular direction. That is, in an ideally specular transfer, incoming light in a single direction produces light in a single outgoing direction—in the *specular direction* $\vec{\omega_{io}}$. In terms of the methods developed in Section 4.1, the

light transport equation for ideally specular transfers can be expressed as a convolution of the incoming light L_i with a delta function in directional space (in the domain of the light transfer problem):

$$L_o(\mathbf{x}, \vec{\omega_o}) = L_e(\mathbf{x}, \vec{\omega_o}) + L_i(\mathbf{x}, \vec{\omega_i}) \star \delta(\vec{\omega_{io}}). \tag{6.1}$$

Convolving any function with a delta represents the sampling of that function in the particular domain location specified by the delta, according to the sifting property of the delta function (Section 2.2) and sampling theory (Section 2.6.1). That is, the convolution term in the equation above is just the sampling of the incoming light function in the specular direction $\vec{\omega_{io}}$ determined by the BDF delta function. Instead of using convolution with a delta function, the equation can be simplified and explicitly written in terms of the sifting domain location, i.e., in terms of the specular direction $\vec{\omega_{io}}$:

$$L_o(\mathbf{x}, \vec{\omega_o}) = L_e(\mathbf{x}, \vec{\omega_o}) + L_i(\mathbf{x}, \vec{\omega_{io}}). \tag{6.2}$$

The integration of the incoming light over direction in Equation (4.1) is unnecessary for ideally specular transfers; the outgoing light from a point in a given direction depends on a single incoming direction of light.

This dependence of the outgoing direction of light upon only the specular direction of incoming light can be seen in terms of a mapping between two spaces. Incoming directions are mapped into outgoing directions and vice-versa.

6.2 A Mapping Approach to Reflections

Abstractly, a mirror surface can be seen as a window to a reflected scene. Each surface point in a scene has its corresponding point in the reflected scene, given by the straight reflection through the mirror surface point. The reflected scene depends on the geometry of the scene, on the curvature of the mirror surface, and on the relative position and orientation of the mirror with respect to the viewer.

A reflection can be seen as a mapping operation. There are two spaces: the space where the scene is defined and the space where the reflected scene is represented (Figure 6.1). In fact, for reflections, we are not directly interested in the reflected space; we are interested in projections of that reflected space onto the mirror surface. The reflected image space is a subset of the reflected space, where the subseting function is perspective projection. The reflected image space requires the definition of a viewer, which is not required for the reflected space. In terms of the mapping, we are interested in the function that maps points between the scene and the reflected image. As for any other mapping operation, there are two alternative mappings for reflection: feed-backward mapping and feed-forward mapping.

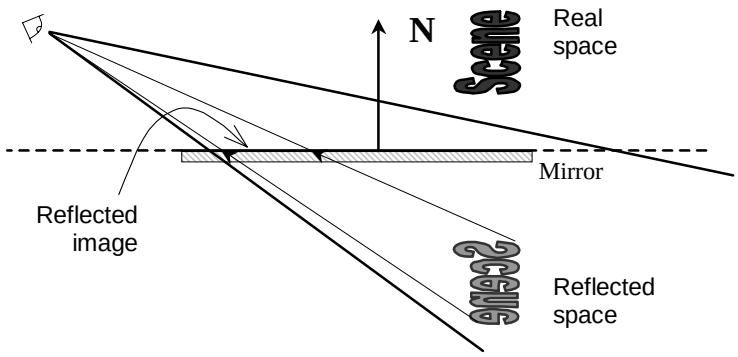


Figure 6.1: *The spaces involved in a reflection on a planar mirror: the real space, the reflected space, and the reflected image or projected reflected space (on the surface of the mirror).*

In a feed-backward approach—i.e., querying the reflected scene for each visible point in the real scene (ray tracing)—there is a simple mechanism to find the reflected image on a mirror surface. For each viewing ray, the visible surface point in the direction of the ray and the normal vector at that point are known; this allows computing the reflected vector at that point and finding the corresponding visible surface point as seen from the reflection point in the reflected direction.

In the feed-forward approach—i.e., mapping points from the real scene onto the reflected image on the mirror surface—a simple mechanism for computing reflections exists only for planar mirrors. For each point in the real scene, find its corresponding reflected location through the mirror plane (Section 6.3); the visible portion of the reflected scene through the mirror is the reflected image. For curved surfaces, though, there is no single reflection plane to map points from the real scene into the reflected scene. A curved surface contains infinitely many tangent (reflection) planes, and one cannot know which one to use for reflecting a point from the real scene into the reflected scene.

The difference between the two alternative mappings for curved surfaces is that the feed-backward approach defines a one-to-one mapping, whereas the feed-forward defines a one-to-many mapping. Clearly, the feed-forward approach is not theoretically appropriate for handling reflections on curved surfaces. However, the feed-forward approach defines a one-to-one mapping for planar mirrors.

6.3 Planar Mirrors

Abstractly, a planar mirror defines a window to the reflected scene (Figure 6.2)—a frustum defined by the viewer and the edges of the mirror to an imaginary representation of the scene reflected with respect to the mirror, or, alternatively, the frustum defined by the reflected viewer and the edges of the mirror into the scene. The solid lines in the figure show the frustum defined by the viewer and the mirror

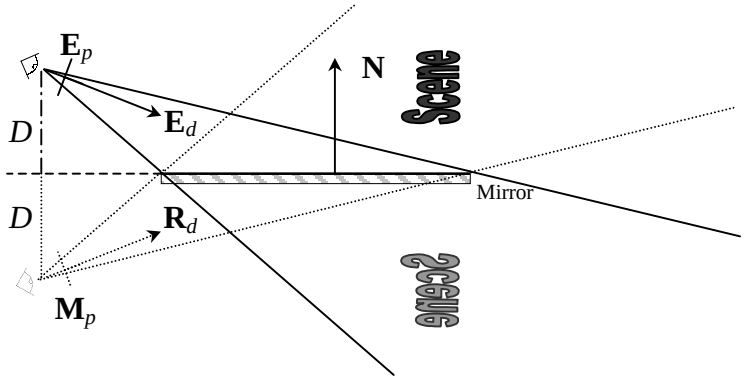


Figure 6.2: *Reflection of the eye-point* $\mathbf{E}_p$ *and the eye-direction* $\mathbf{E}_d$ *with respect to a planar mirror.*

edges. The dashed lines show the reflected frustum defined by the reflected viewer and the mirror edges. For a planar mirror, the reflected image is the planar perspective projection of the reflected scene onto the planar mirror surface (Figure 6.3).

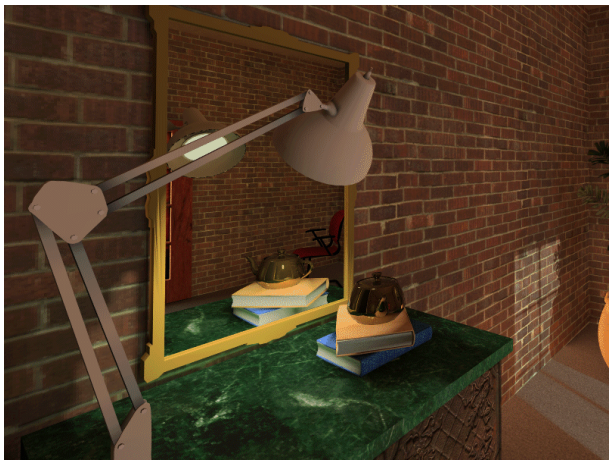


Figure 6.3: *A planar mirror illustrating the planar perspective projection on the reflected image. Notice the perspective shortening on the reflected brick wall.*

Algorithmically, a scene with a single mirror can be drawn in two alternative forms: by double modeling or with a two-pass approach. The double modeling solution actually models the reflected scene in its physical location with respect to the scene and draws both the scene and its reflected version with a single pass. The two-pass approach is split as follows. In the first pass, the reflected image of the scene is drawn. In the second pass, the rest of the scene (non-mirror surfaces) is drawn. The first pass reflects the eye-point $\mathbf{E}_p$ and the eye-direction $\mathbf{E}_d$ with respect to the plane supporting the mirror

(Figure 6.2) and draws the scene from the mirrored view pose. The mirrored eye-point $\mathbf{M}_p$ and the mirrored eye-direction $\mathbf{R}_d$ are given by

$$\mathbf{M}_p = \mathbf{E}_p + 2D(-\mathbf{N}) \tag{6.3}$$

and

$$\mathbf{R}_d = 2(\mathbf{N} \cdot \mathbf{E}_d)\mathbf{N} - \mathbf{E}_d. \tag{6.4}$$

where D is the shortest distance from the eye-point $\mathbf{E}_p$ to the plane, $\mathbf{N}$ is the normal vector to the mirror, and $\mathbf{E}_d$ is the eye-direction. In the second pass, the scene is drawn from the original view pose but without drawing the mirror surface. Alternatively, the second pass can use a texture-mapping technique: the reflected image computed in the first pass is applied as a texture onto the surface of the mirror which must then be drawn with the texture in the second pass. The texture-mapping approach has the additional burden in current graphics architectures of requiring the transfer of the reflected image from the color buffer into texture memory per frame.

The description above assumed either a single planar mirror in the scene or that we are not handling interreflections when there is more than one mirror in the scene. If there are two or more mirrors in the scene, the same algorithm has to be applied recursively. In the first pass for each mirror in the scene, determine if the other mirrors are visible from the respective reflected view poses. If so, apply the algorithm recursively to each of the visible mirrors. The recursion terminates when one of the mirrors does not "see" any of the other mirrors from the corresponding reflected view pose or when the recursion reaches a pre-imposed recursion limit.

Although this technique for rendering reflections on planar mirrors is simple, it is intensive in terms of computational time. Each planar mirror in the scene requires the rendering of the entire scene from its mirrored view pose for each frame. This means dealing with the entire complexity of the scene several times every frame, which is impractical for reasonably complex scenes. The case with multiple recursive mirror reflections is even harder; the number of re-renderings of the entire scene per frame grows exponentially with the number of mirrors in the scene and with the number of recursion levels performed. Consider the limiting case of parallel facing mirrors as in the Hall of Mirrors at Versailles. The recursion process is unavoidable for rendering mutually visible specular reflectors. Thus, interactive rendering of planar mirror reflections demands more efficient scene rendering at each level of the recursion than does rendering unmirrored scenes.

6.4 Optimized Scene Rendering

Interactive rendering of large models can easily exceed the performance of graphics hardware systems; the task becomes even harder when the model needs to be rendered several times per frame, as for reflections. Often there are more primitives to be rendered than pixels on the screen. Moreover, in general more than one point in the scene maps to the same pixel in an image (just the closest point is kept). These two observations imply that the rendering engine is handling more data than a final image can represent. This implies a waste of computational time—and leads us to the UNC Walkthrough Project motto:

> *Avoid putting into the rendering pipeline polygons that you cannot finally see.*

The fundamental idea is to decouple rendering time from scene complexity. An analysis of what happens when visualizing a geometrical model indicates a few approaches for controlling the number of primitives that have to be sent to the graphics pipeline. Primitives should not be sent to the graphics pipeline if:

- they are outside of the view frustum (*view-frustum culling*)
- they face away from the viewer (*back-face culling*)
- their projection on the image is substantially smaller than a pixel (*model simplification*)
- they are behind another opaque object (*occlusion culling*)

Software techniques exist for handling the situations above and their names appear in parentheses. The interested reader is referred to [Aliaga99] for a description of a system that combines all the techniques above and provides references to earlier work on each of the topics. This chapter draws upon the fourth approach above for accelerating scene rendering in mirror reflections. An image-based representation of the scene taken from selected view poses is precomputed and then warped at runtime to compute new images from novel view poses. An image of a scene captures only the first layer of visible points for the particular view pose, i.e.; a warping of such an image does not deal with the occluded regions from the original view pose.

6.5 Image-Based Rendering

Recently, image-based methods have emerged as possible tools for producing views of an environment in time that is independent of scene complexity [Chen93, McMillan95, Levoy96, Gortler96]. The idea is to precompute images from several view poses in the scene and then derive, from the set of precomputed images, new images for arbitrary view poses at runtime.

We follow the method in [McMillan97]. A set of images-with-depth[1] is precomputed for a scene and then reprojected at runtime to derive new images. The precomputation represents a point sampling of the environment—each pixel captures the three-dimensional location and color of the first visible surface along the pixel direction as seen from the COP (Figure 6.4)—and the runtime step is a reprojection of those 3D point samples to reconstruct images for novel view poses. Note how

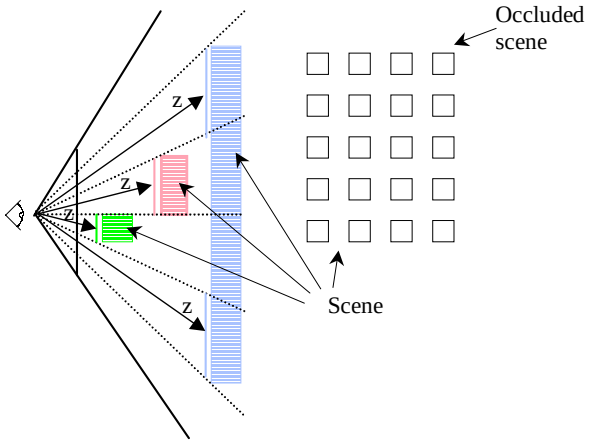


Figure 6.4: *An image-with-depth of a simple scene: each arrow labeled with a 'z' represents a pixel surrounded by dotted lines representing the corresponding solid angle. Notice that the occluded part of scene (solid line squares) is not captured.*

the precomputation step reduces depth complexity to a value of one on any image-with-depth of an arbitrary scene (assuming that an image captures a single color and a single depth value per pixel). Clearly, the depth complexity reduction of this method can be notable. However, this depth complexity reduction does not always translate directly into efficiency. The point sampling of the scene generates larger amounts of data than the original scene composed of polygons, and image reprojection suffers from artifacts.

[1] An *image-with-depth* captures color and depth of the first visible surface along each pixel direction as seen from the COP. Therefore, each pixel represents a point in three-dimensional space with its corresponding color. An image-with-depth is a point sampling of a scene composed of continuous surfaces from a particular view pose.

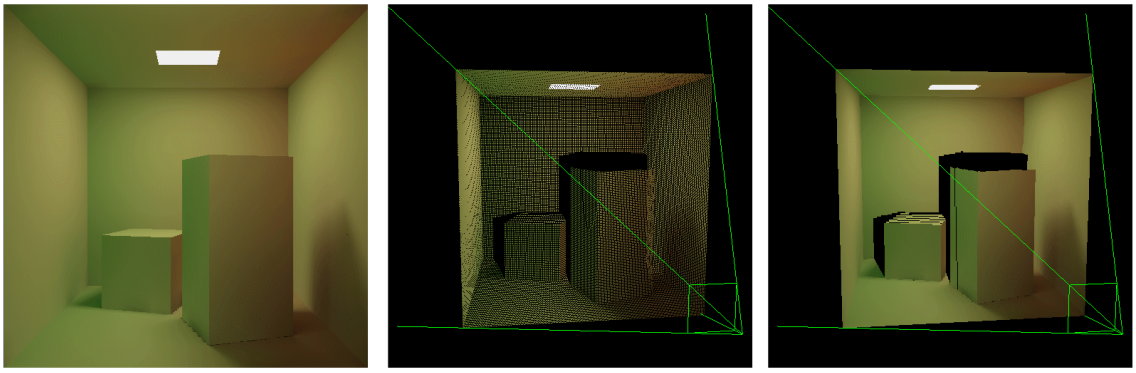


Figure 6.5: *Image-with-depth example: (left) original scene, (middle) the reprojection of a single image-with-depth showing its view-frustum, and (right) the same image-with-depth with reconstruction.*

The large amounts of data produced by the image-based approach may require memory management and data fetching techniques when the image data does not all fit in memory [Aliaga99]. As for artifacts, the reprojection of images-with-depth is prone to disocclusion artifacts. *Disocclusion* or *exposure* artifacts appear in the final images as holes with no color data due to the invisibility of those spots from the original view poses. For example, let's suppose a single image-with-depth that captures points only on a face of a cube (Figure 6.6). Now suppose a new view pose that would uncover another face of the cube (Figure 6.7). A reprojection of the image-with-depth to the new view pose will present a hole in the region of the uncovered face of the cube, since that part of the model was not sampled by the image-with-depth. Notice the disocclusion artifacts in the reprojections of a single image-with-depth in Figure 6.5.

6.6 Image-Based Approach to Planar Mirror Reflections

We compute mirror reflections on planar surfaces by using the image-based rendering approach. Since the amount of computation required to reproject an image is proportional to the image size, the time required to render a mirrored view of the scene is constant in the resolution of the images. This is in contrast to geometry-based rendering of reflections (Section 6.3), whose time depends on the geometric complexity of the model. We deal with disocclusion artifacts by rendering multiple images-with-depth taken from view poses near the desired view pose. Since distinct view poses capture different visible regions of the scene, rendering more than a single image-with-depth decreases disocclusion artifacts.

The algorithm starts by precomputing visibility for each planar reflector in the scene. For each reflective object, a set of images-with-depth is precomputed for points behind the mirror surface

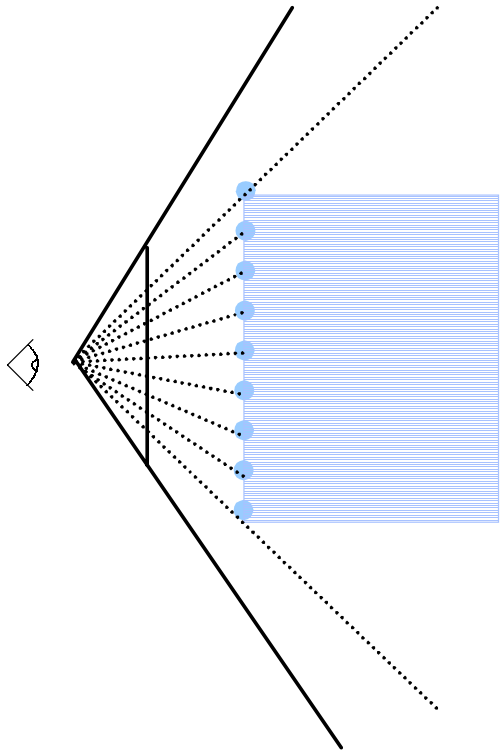


Figure 6.6: *Image-with-depth of a cube face: notice that the image-with-depth captures information only about one of the faces of the cube—the first visible surface along the direction of each pixel.*

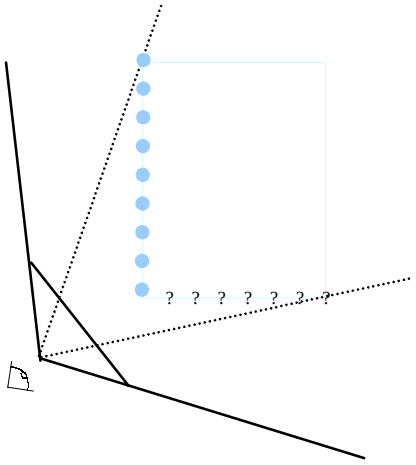


Figure 6.7: *Image-with-depth reprojected to a new view pose: notice the lack of information to reconstruct the uncovered face of the cube.*

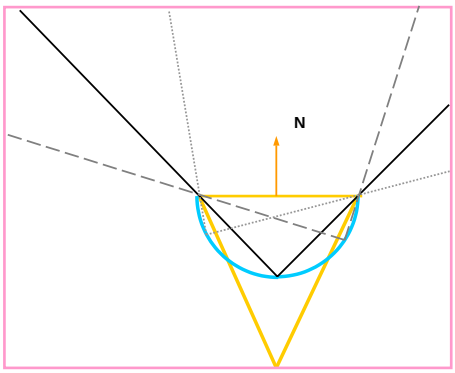


Figure 6.8: *Cross-section of a simple scene containing a pyramid inside a cube. The surface indicated with a normal vector—the base of the pyramid—is a mirror. The hemisphere behind the mirror serves for placing images-with-depth for capturing what is visible from the surface of the mirror. Three distinct view frusta are shown with three different line styles.*

(Figure 6.8). The points are uniformly distributed on a hemisphere behind the reflector and define the centers of projection for the images-with-depth. Each COP and the edges of the mirror define a view frustum for an image-with-depth. The set of images-with-depth represents what is seen from the reflector's surface over a range of angles and positions. In fact, the set of all pixels in the images-with-depth represents a point-sampled version of the original model. A detail worth noticing is that the point-sampling captures only the first layer of surfaces visible from the mirror surface; for example, the images-with-depth of a mirror inside a room with opaque walls do not point-sample objects outside of the room, which may contain higher complexity than the objects inside the room. This capturing of just the first visible layer of surfaces (precomputation of visibility/occlusion from the mirror point of view) translates into rendering speed-ups.

The image reprojection approach assumes that all sampled points in a scene emit light isotropically, so that light intensity (color) information stored in images-with-depth from a given view pose is valid for any reprojected view pose. That is, images-with-depth have to store view-independent light information, in order to produce valid reprojections at arbitrary view poses. Our images-with-depth sample radiosity-illuminated models (Chapter 5), so that the color information stored by the images is a view-independent quantity.

Our image-based approach for rendering reflections on planar surfaces has four distinct steps: sampling, selection, reprojection, and reconstruction. The first step is part of a preprocessing phase, whereas the last three are performed at runtime. The result of this image-based pass is a perspectively-correct reconstructed view of the environment as seen from a mirrored viewpoint. The next sections discuss the four steps above.

6.6.1 Sampling

The sampling step captures the regions of the scene visible from the surface of the mirror. A set of images-with-depth is created for each planar mirror (figures 6.8 and 6.9). The sampling process

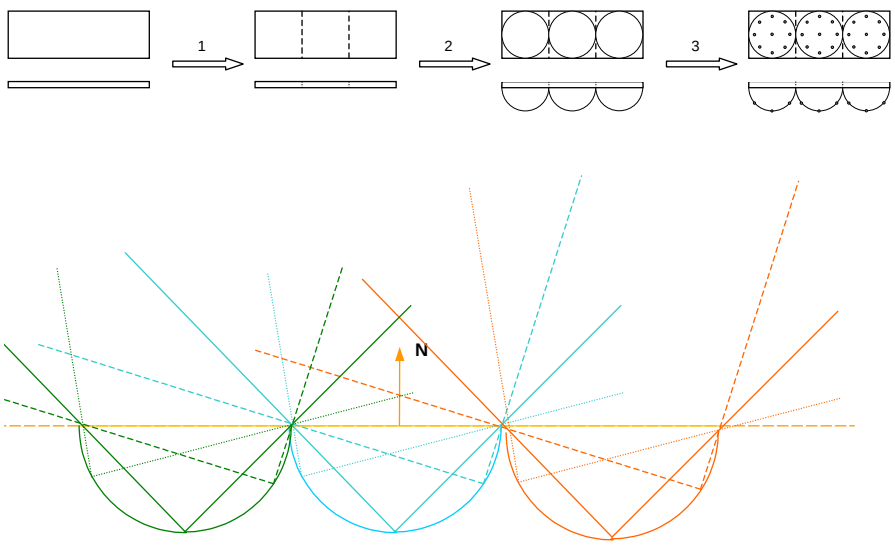


Figure 6.9: *Sampling step: the top figure illustrates the three sampling stages on a top view and on a side view of a planar reflector; the bottom figure shows the distribution of view-frusta for images-with-depth for a planar reflector.*

proceeds in three hierarchical stages to finely sample what is visible from the surface of the reflector. First, the planar reflector is discretized at a given planar resolution. Each of the resulting elements on the plane defines a reflector element on the mirror object. Then, a hemisphere is placed centered at the back side of each element; the base of the hemisphere is contained inside the reflector element. Finally, points are uniformly distributed on the surface of the hemisphere. Each point on the hemisphere defines the COP for an image-with-depth. The (possibly off-axis) view-frustum for each image-with-depth is created using one point on the hemisphere as the center-of-projection and the vertices of the reflector element as the near plane. Both the density of hemispheres and the density of points on the surface of each hemisphere depend on the distance from the mirror to the reflected scene. More discrete elements, and hence more hemispheres, and more points on each hemisphere, and hence more images-with-depth for each discrete element, are needed as the reflected scene is closer to the mirror surface. Figure 6.8 shows how the locations of the images-with-depth are chosen for a single reflector element, whereas Figure 6.9 shows the distribution of images-with-depth for a reflector discretized into three reflector elements.

In terms of implementation, the Class 6.10 summarizes what is stored for each image-with-depth. In that class, `nCols` and `nRows` define the number of columns and rows of the image, `color` and

```
class ImageWithDepth {
   int             nCols, nRows;
   unsigned char * color;
   float *         depth;
   Mat44           inverse;
}
```

`Class 6.10:` *Data structure for holding image-with-depth information.*

`depth` contain the color and depth information per pixel for the image, and `inverse` represents the inverse of the geometrical transformation used to place the virtual camera in the reflected view pose when computing the image-with-depth. The pseudo-code in Program 6.11 describes how to compute an image-with-depth.

```
Setup camera pose (modelviewMatrix, projectionMatrix, viewportMatrix)
Render scene
Save inverse of camera pose:
   inverse =  inverse_modelviewMatrix*inverse_projectionMatrix*inverse_viewportMatrix
Save color buffer and depth buffer
```

`Program 6.11:` *Computing an image-with-depth.*

Note a crucial difference between a traditional image (2D projection) and an image-with-depth; a traditional image captures the projected scene onto the screen plane, whereas an image-with-depth captures the location of intersected points by pixel rays in 3D space. This observation implies that all the pixels in a traditional image have the same size (the pixel area on the screen plane), whereas, for perspective projection each pixel in an image-with-depth can potentially have a different size based on their distance to the viewer. Each pixel represents visibility inside a small pixel frustum and the farther from the COP the pixel ray intersection with an object occurs, the larger the surface area subtended by the pixel frustum. To distinguish between traditional and image-with-depth picture elements, we refer to image-with-depth picture elements as *points*.

6.6.2 *Selection*

The sampling step used a set of images-with-depth to capture the visible regions of the model from a range of positions and orientations from behind the planar mirror. At runtime, the selection step finds the image(s) in the set of each mirror that best approximate the corresponding reflected image.

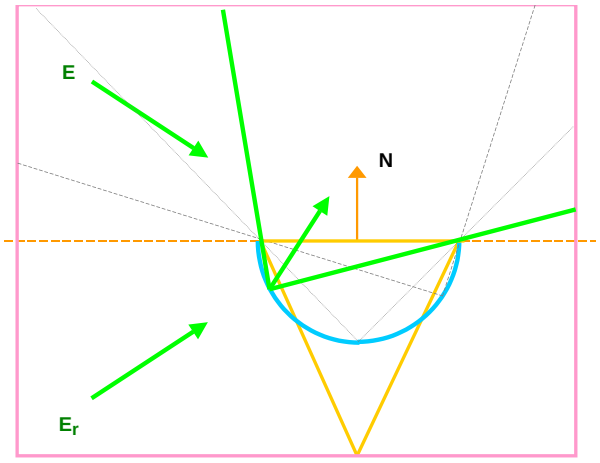


Figure 6.12: *Selection step.*

The selection step mirrors the viewpoint and the viewing direction about each reflection plane (Section 6.3) and selects the image(s) that are closest to the mirrored view pose (Figure 6.12). By examining the viewing angle formed by the mirrored view direction and the viewing direction of the images-with-depth, the selection determines which images-with-depth have the highest image resolution in the reflected view pose. This is a simple selection process implemented as a search for the minimal angle between the mirrored view direction and the view directions of the images-with-depth. A distance-based selection technique is also possible: based on the distance from the COP of the precomputed images-with-depth to the COP of the reflected camera, the algorithm selects the closest one(s). Both selection methods perform equally well when the reflected view pose is at a distance much greater than the diameter of the hemisphere. However, when the reflected view pose is close or inside the hemisphere, the directional method provides better results.

6.6.3 *Reprojection*

The selection step found the image(s) with COP(s) closest to the reflected view pose in the set of precomputed images-with-depth of the planar mirror. However, the selected image(s) were potentially not computed from the same view pose as the reflected runtime view pose, because either the exact

direction or the exact COP was not sampled. The precomputed images-with-depth need to be warped to the new reflected view pose to approximate the visible regions of the model from that new view pose.

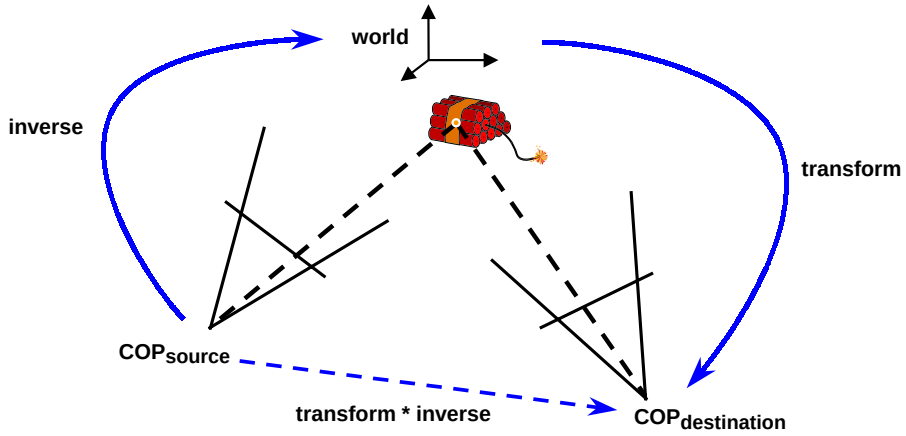


Figure 6.13: *Image-with-depth reprojection.*

To warp the precomputed images to the new view pose, the pixels of the images-with-depth can be projected back to world space by using the inverse of the transformation matrix used to compute the images-with-depth (Figure 6.13). Then, the points in world space are transformed into the new reflected view pose using the current transformation. This process represents the reprojection of points between two three-dimensional spaces. Our implementation exploits the graphics hardware in performing the reprojection, and depth is resolved by the Z-buffer when two or more points map to the same pixel in the reflected image. Each pixel in an image-with-depth is dealt with as a point in three-dimensional space that is sent through the regular transformation pipeline. The transformation matrix is preloaded with the product of the current runtime transformation and the inverse transformation of the respective image-with-depth. The resulting transformation matrix takes points from the image-with-depth to the current view pose. In terms of OpenGL implementation, pixels from an image-with-depth are reprojected using the pseudo-code in Program 6.14 (along with the class definition of an ImageWithDepth (Class 6.10)).

6.6.4 Reconstruction

A reconstruction process is necessary to produce an image from the possibly scattered points on the screen resulting from the reprojection step (Figure 6.5). The reprojection simply maps 3D locations from the original eye-space where the images-with-depth were computed into the new eye-space of the reflected view pose. Nothing ensures that the reprojected points from the original images-with-

```
Setup new camera pose (modelviewMatrix, projectionMatrix, viewportMatrix)
current_transform *= imageWithDepth->inverse
glBegin( GL_POINTS );
      For x=0 to imageWithDepth->nCols
            For y=0 to imageWithDepth->nRows
                  glColor(imageWithDepth->color[x,y]);
                  glVertex(x,y,imageWithDepth->depth[x,y]);
glEnd();
```

Program 6.14: *Reprojecting an image-with-depth.*

depth cover all the pixels in the final image; disocclusion artifacts need to be avoided, which we do by reprojecting multiple images-with-depth taken from view pose around the desired view pose. In addition, from our definition of image-with-depth, a point is actually an area element, which implies that a reprojected point has a variable size depending on the original and the new view poses. This variable point size indicates that each projected point on the new screen may cover multiple pixels.

When reprojecting an image, two situations can arise: the magnification or the minification of the original picture elements into the new image. Magnification happens when the destination space has higher spatial resolution than the source space, whereas minification happens when the destination space has lower spatial resolution than the source space. Magnification needs to create data in between pixels, whereas minification needs to filter out data. Our reconstruction works in conjunction with the reprojection step. Magnification is done by controlling the size of reprojected points on the destination screen space (the pixel area) and minification is performed by skipping pixels in the original image when reprojecting and reconstructing an image-with-depth. Program 6.15 shows how to extend Program 6.14 to perform magnification. Note that Program 6.14 from the previous section assumed

```
Setup new camera pose (modelviewMatrix, projectionMatrix, viewportMatrix)
Multiply( current_transform, imageWithDepth->inverse )
For x=0 to imageWithDepth->nCols
      For y=0 to imageWithDepth->nRows
            glPointSize( EstimatePointSize(imageWithDepth->depth[x,y]) );
            glBegin( GL_POINTS );
                  glColor(imageWithDepth->color[x,y]);
                  glVertex(x,y,imageWithDepth->depth[x,y]);
            glEnd();
```

Program 6.15: *Reprojecting an image-with-depth with per-point magnification.*

a constant point size (projected area) for all the picture elements in an image-with-depth, whereas the pseudo-code in this section sets a new point size for each pixel in an image-with-depth.

For the case of magnification, an estimate of the x and y dimensions in pixels of the projected point size on the screen is given by:

$$pointSize_{x,y} = \frac{1}{step_{x,y}} = \frac{distanceScale \times directional_{x,y}}{window_{x,y}} \tag{6.5}$$

where $window_{x,y}$ is a spatial resolution scaling factor from the source to the destination image in x or y, $distanceScale$ is a distance scaling factor based on the point-to-COP distance on the two spaces, and $directional_{x,y}$ is a directional scaling factor based on the dot product of the desired view-direction and the X and Y axes in the source image-space. The spatial resolution scaling factor makes sure that one unit in the source space gets scaled to the same relative dimension in the destination space, which in terms of image spatial resolutions is

$$window_{x,y} = \frac{SRCresolution_{x,y}}{DESresolution_{x,y}}. \tag{6.6}$$

The distance-scaling factor compensates for perspective shortening. Surfaces intersected closer to the COP have smaller areas than surfaces intersected farther away. The distance-scaling factor is based on the ratio of point-to-COP distances on the two spaces:

$$distanceScale = \frac{SRC_{depth}}{DES_{depth}} \tag{6.7}$$

The directional scaling factor compensates for projected area, assuming that the intersected surface element at each pixel in the source image is a square parallel with the screen. The projected square in the destination space is scaled by the dot product of the desired view-direction and the X and Y axes in the source image-space.

This reconstruction step of just increasing the point size of the projected point on the screen is similar to splatting (Chapter 2) of a box kernel aligned with the screen. The subtle difference between this and splatting is that the reprojection step with resized points does not perform the per-pixel integration of the convolution process characteristic of splatting. Our approach uses Z-buffering, which implies replacement of pixel information instead of accumulation.

Equation 6.5 can also be used for minification of the images-with-depth—when the projected area of the image-with-depth is smaller than its original area. In this case, a step size estimation ($step_{x,y}$) is used to skip points when reprojecting the image-with-depth. The pseudo-code 6.16 performs reprojection and minification. Minification assumes a constant point size for an entire image-with-depth.

```
Setup new camera pose (modelviewMatrix, projectionMatrix, viewportMatrix)
Multiply( current_transform, imageWithDepth->inverse )
stepSize = 1.0 / EstimatePointSize(imageWithDepth->depth[x,y]);
glBegin( GL_POINTS );
      For x=0 to imageWithDepth->nCols step stepSize
            For y=0 to imageWithDepth->nRows step stepSize
                  glColor(imageWithDepth->color[x,y]);
                  glVertex(x,y,imageWithDepth->depth[x,y]);
glEnd();
```

Program 6.16: *Reprojecting an image-with-depth with minification.*

6.6.4.1 A hierarchical—quadtree—organization for images-with-depth

The estimation of point size on a per-point basis for magnification as discussed above clearly generates good results; however, it is a bottleneck. The point size needs to be estimated for each pixel, which is a computational burden. And the estimated point size needs to be sent to the graphics pipeline for each pixel, which increases the amount of data required to transfer the entire image-with-depth to the pipe—besides the three-dimensional location and the color of each point, it is also necessary to send the point size.

To keep the computation reasonable, we organize the points of an image-with-depth in a quadtree. The quadtree subdivision is performed in the XY plane until nodes of the quadtree contain only points in a given range of depths or have a minimum number of points (Figure 6.17). For each

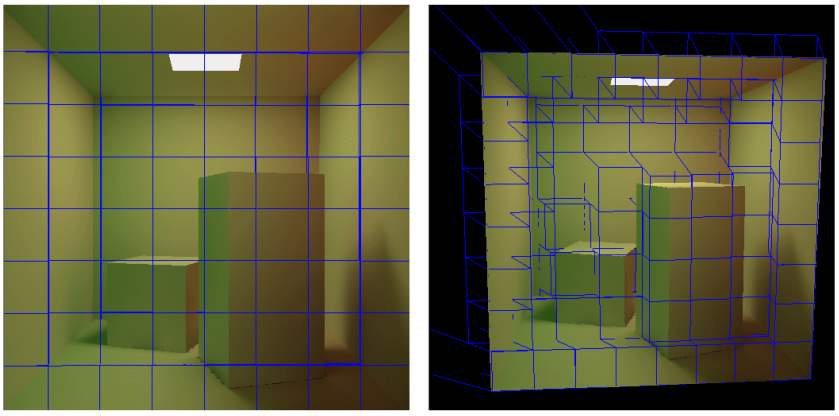


Figure 6.17: *A quadtree organization of an image-with-depth. Notice the different depth ranges of nodes in the quadtree.*

node, a representative point is computed as the average (center of mass) of all the points inside that node. The representative point of a quadtree node is used to estimate the point size for all the points

inside that node (Equation (6.5)). This reduces the amount of computation and bandwidth required by the per-pixel approach.

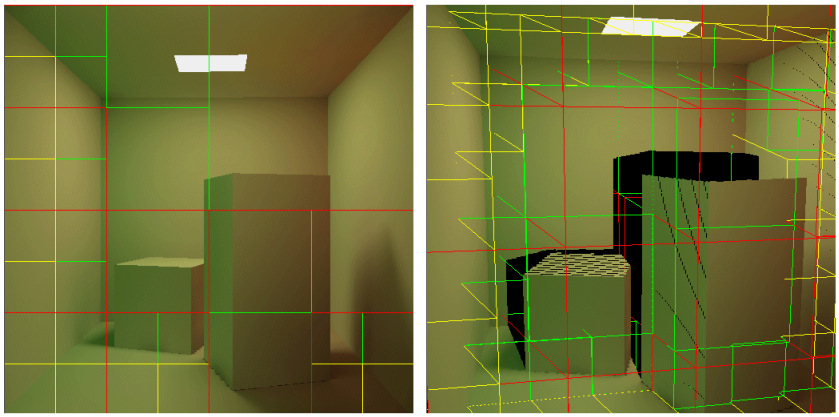


Figure 6.18: *View-frustum culling the nodes of a quadtree organization of an image-with-depth. Quadtree nodes that are completely inside the frustum are drawn in green, whereas nodes that are partially inside are drawn in yellow.*

In addition, this hierarchical approach permits view-frustum culling of sub-regions of the images-with-depth (Figure 6.18). A view-frustum culling traversal of the quadtree can eliminate quadtree nodes (and all the associated points) that are outside of the frustum.

6.6.5 Results for Planar Mirror Reflections

Figure 6.19 shows the rendering of image-based reflections for the floor and for the piano top of a room of a house model, which has $140,000$ polygons and five planar mirrors sampled as described in Table 6.6.5. The image-based reflections used two images-with-depth per hemisphere for each mirror. Notice the disocclusion artifacts on some regions on the floor and on the piano top nearer to the viewer. Notice also the noise introduced in the reflected images due to the zero-order quasi-splatting reconstruction scheme used for producing the image-based reflections. This noise is clearly undesirable for mirror reflections, but will show useful for glossy reflections produced by blurring mirror-reflected images (next chapter). The average performance for rendering this scene was 15 frames per second on an SGI Onyx2 workstation using a single 250MHz R10000 processor and an InfiniteReality2 graphics pipe with four raster managers.

We analyzed the performance of image-based reflections with the house model. Images were rendered at 720×486 and performance data were collected by playing a pre-recorded path (944

Figure 6.19: *Image-based reflections on the floor and on the piano top using two images-with-depth per hemisphere for each reflector.*

Name	*# of hemispheres*	*Images-with-depth per hemisphere*	*Image-with-depth resolution*	*Memory (Mbytes)*
Piano top	1	23	128×128	2.54
Living floor	12	6	128×128	7.98
Door mirrors	11	11	128×128	20.69
Music floor	8	7	128×128	8.21
Bench top	1	67	32×32	0.57

Table 6.1: *Data for the five reflectors in the house model.*

frames) through the model. Table 6.6.5 describes the data for the images-with-depth for each of the five reflectors in the scene.

The graph in Figure 6.20 shows the performance for our image-based approach versus the geometry-based approach. It illustrates the cost of increasing the number of images-with-depth reprojected per reflector. Notice that the image-based approach with one or two images was twice as fast as the geometry-based one. Note that increasing the number of selected images-with-depth improves the quality of reflected images.

6.7 Directionally Dependent Mirror-Like Reflectance

The approach for mirror reflections discussed so far assumed ideal reflectance of the material: the incoming light from a certain direction is completely transferred to the outgoing direction. However, most polished materials do not exhibit this direction-independent reflectance behavior. Most materials exhibit higher reflectance values at grazing angles than at normal incidence. In addition, some materials filter the color of the reflected light in a directionally dependent manner. Both effects are

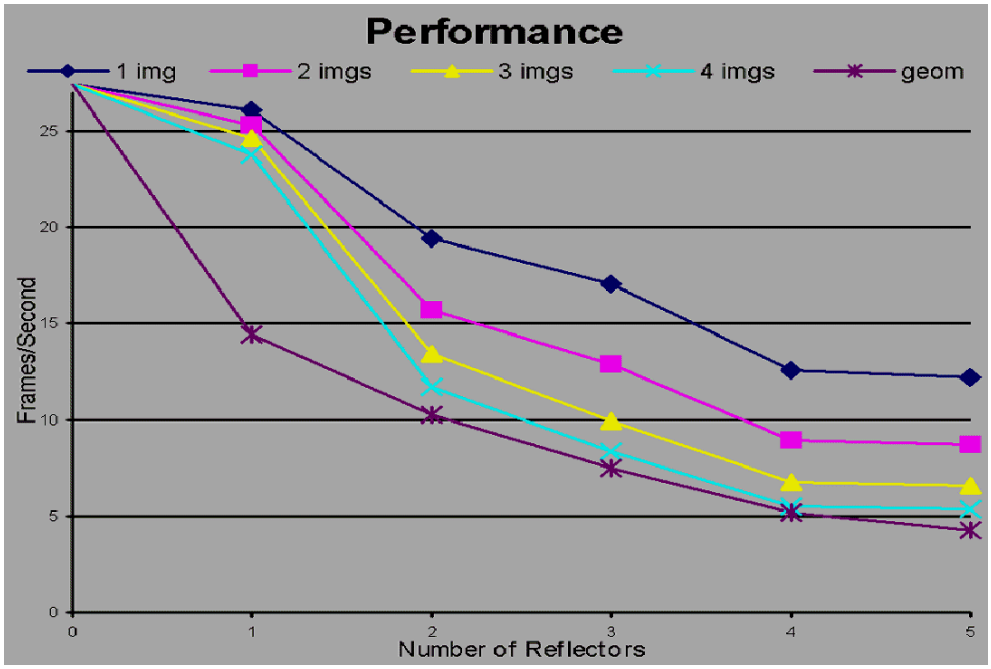


Figure 6.20: *Performance comparison of image-based reflections with varying number of images-with-depth per reflector with respect to geometry-based reflections.*

produced by directionally dependent variations of material reflectance. The first effect is produced by equal reflectance variations for all wavelengths of light, whereas the second effect is due to unequal reflectance throughout the spectrum of light. We call the materials that exhibit this directionally dependent reflectance *mirror-like* materials. This name distinguishes them from ideal mirrors, which produce ideally specular reflections.

Most materials exhibit higher reflectance values at grazing angles than at normal incidence [Cook81, Feynman89]. Notice, for example, how the intensity of a specular reflection varies on planar glass: looking at a grazing angle with the glass you should see a much brighter reflection than at normal incidence. This same behavior takes place on polished surfaces made of materials usually not considered specular. As an extreme case, observe the specular reflection of a bright light source on a sheet of paper at a very grazing reflection angle. Other interesting and not so extreme examples of this type of reflectance behavior are tiles on bathroom walls and clearcoat on automobiles.

Some materials filter the color of the reflected light in a directionally dependent manner. For example, notice the reddish reflections at grazing angles with copper surfaces. Gold also presents a peculiar color shift for reflections: reflections at grazing angles do not get color shifted, but at near grazing angles reflections are golden colored. This behavior is a characteristic of all metals [Cook81, Feynman89].

The directional-dependent or mirror-like reflectance affects a mirror reflection only by performing a color filtering of the light in each outgoing direction. In terms of the light transport equation for ideally specular transfers (Equation (6.2)), the directionally dependent reflectance implies a reflectance function modulating the light transferred term:

$$L_o(\mathbf{x}, \vec{\omega_o}) = L_e(\mathbf{x}, \vec{\omega_o}) + \rho_{bd}(\mathbf{x}, \vec{\omega_i}, \vec{\omega_o}) L_i(\mathbf{x}, \vec{\omega_{io}}) \tag{6.8}$$

where $\rho_{bd}(\mathbf{x}, \vec{\omega_i}, \vec{\omega_o})$ is the bidirectional reflectance distribution function of the material at point $\mathbf{x}$ and at incoming and outgoing directions $\vec{\omega_i}$ and $\vec{\omega_o}$.

6.7.1 Approximating Directionally Dependent Reflectance

The directionally-dependent reflectance is due to the roughness of the surface and the shadowing and masking of incident and reflected light (Section 3.5.3). Although surface roughness can cause scattering of light, which would be out of the scope for this chapter, this section considers only the directional variation of reflectance without taking into account the blurriness produced by scattering. Inspired by Cook and Torrance [Cook81], we model directionally dependent reflectance with the following expression:

$$\rho = \frac{F}{\pi(\mathbf{N} \cdot \mathbf{L})(\mathbf{N} \cdot \mathbf{V})} \tag{6.9}$$

where $\mathbf{N}$ is the normal vector at a point in question illuminated from direction $\mathbf{L}$ and viewed from direction $\mathbf{V}$. The scalar F is a reflectance term derived from the Fresnel equation [Ditchburn91, Fowles89, Feynman89] that expresses the reflectance of a perfectly smooth mirror-like surface in terms of the wavelength of the incident light, the geometry of the surface and the light, and the angle of incidence. To make it practical, instead of using the general equation, we follow the compromise presented by Cook and Torrance [Cook81] and well reviewed in [Glassner95, Watt92]. We fit the Fresnel equation to the measured normal reflectance for a polished surface (normal reflectance is known for most materials). The reflectance F is then given by

$$F = \frac{1}{2}\frac{(g-c)^2}{(g+c)^2}\left(1 + \frac{(c(g+c)-1)^2}{(c(g-c)+1)^2}\right) \tag{6.10}$$

where

$$c = \cos(\theta_i) = \mathbf{V} \cdot \mathbf{H}, \;\; \mathbf{H} = \frac{\mathbf{L} + \mathbf{V}}{2} \tag{6.11}$$

and

$$g^2 = n^2 + c^2 - 1. \tag{6.12}$$

The index of refraction n of the material needed for Equation (6.12) is usually unknown—a function of direction and wavelength of incident light. However, reflectance for normal incidence is usually known, $\theta_i = 0$, which let us approximate an effective index of refraction for the material:

$$F_0 = \left(\frac{n-1}{n+1}\right)^2 \tag{6.13}$$

which solving for n gives

$$n = \frac{1+\sqrt{F_0}}{1-\sqrt{F_0}}. \tag{6.14}$$

The values of the refractive index are then substituted into Equation (6.10) to compute the reflectance F for any angle of incidence. This procedure is wavelength-dependent, since the reflectance at normal incidence is a wavelength-dependent quantity, which implies that it has to be applied to the three (RGB) color bands to provide F_r, F_g, and F_b—the color band reflectances of the material.

In summary, the directionally dependent reflectance is represented with functions derived from the reflectance of materials at normal incidence of light. The color shift is produced when the color band reflectances have distinct behaviors for a given material.

6.7.2 A Texture-Mapping Approach to Directionally Dependent Reflectance

The directionally dependent reflectance from Equation 6.9 is a function of the viewing vector, the surface normal, the material refractive index, and the light wavelength. Except for the viewing vector, the other parameters will usually be constant from frame to frame. This suggests look up into a table indexed by the viewing vector, i.e., a view-dependent table look up technique. This view-dependent technique has to modulate the reflected light from a mirror reflection.

In terms of implementation, we can follow a two-pass approach. In the first pass, a mirror reflection on the mirror-like surface is rendered. In the second pass, we process (modulate) the mirror image from the first pass to take into account the directionally dependent reflectance. Note that the first pass involves potentially dealing with the whole complexity of the scene to compute the mirror reflected scene, whereas the second pass has to consider only the mirror-like object (a single polygon for a planar mirror). Note also that mirror-reflected images for the first pass can be computed with any of several techniques such as ray tracing, environment mapping, and the techniques described in Section 6.3 and

Section 6.6. The second pass depends only on the mirror-reflected image stored in the frame buffer and on the geometry and material properties of the mirror-like surface, i.e., it applies for any of the mirror reflection techniques. After the mirror reflection is in the frame buffer, the second pass then renders the mirror-like surface (a single polygon for a planar mirror) on top of the mirror reflected image and modulates the pixel values contained in that region. We implement this color modulation using a blending operation along with a texture-mapping operation; the blending operation is multiplicative blending and the texture-mapping technique is sphere mapping.

Sphere mapping is the only view-dependent texture-mapping operation available in the current graphics hardware [Zimmons99] (Figure 6.21)—it is a view-dependent texture lookup in hardware. Sphere mapping operates in a subset of spherical coordinates—a projection in spherical coordinates, which disregards the spherical radius parameter. A *sphere map* is a planar representation of the two spherical angles into polar coordinates of circular area. The circular map is implemented as a square texture where texels outside an inscribed circle are never accessed and texels inside the disk contain useful values. The center of the disk represents the origin of spherical coordinates. The radius represents the θ angle of spherical coordinates and the angle around the origin represents the angle ϕ of spherical coordinates. The view-dependence in sphere mapping is taken into account by indexing the texture map based on reflected viewing directions. Sphere mapping evaluates the reflected viewing vector at the vertices of a surface and uses that vector to derive the texture coordinates and to index into a sphere map [Zimmons99, McReynolds98, Woo96]. Viewing vectors anti-parallel to the surface normal at a vertex are mapped to the center of the disk and viewing vectors perpendicular to the surface normal map to the outer ring in the disk.

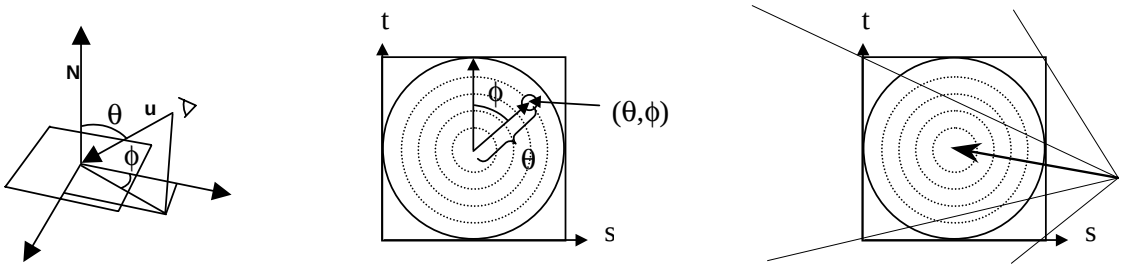


Figure 6.21: *Sphere mapping.*

We store view-dependent reflectance information in sphere maps, based on the material properties of reflectors and on the geometry imposed by the sphere mapping technique. For each texel in a sphere map, there is a viewing direction and a corresponding reflected direction [McReynolds98].

These two directions are used as the incoming and outgoing directions for sampling the reflectance function to be stored at a texel. The reflectance function is either the function described in Section 6.7.1 or any other BRDF. The code in Program 6.22 shows how the reflectance maps are precomputed for the Cook and Torrance reflectance. Note that even though we have chosen to sample the Fresnel equation,

```
// Compute the reflectance map
Image * map = new Image(texelsOnSide, texelsOnSide);
Image * texel = map;
for ( int ns=0; ns< texelsOnSide; ns++ ) {
   for ( int nt=0; nt<texelsOnSide; nt++ ) {
      mapping( texelsOnSide, (float)ns, (float)nt, &x, &y );
      if ( sqrt(x*x + y*y) > 1.0 ) {    // if outside the unit disc
         reflectanceMap[ texel++ ] = 255 * maxR;
         reflectanceMap[ texel++ ] = 255 * maxG;
         reflectanceMap[ texel++ ] = 255 * maxB;
      }
      else {
         Vec3f N( x, y, sqrt(1-x*x-y*y) ); // compute normal vector N
         N.Normalize();
         Vec3f R = U - 2*(U dot N) * N;        // compute reflected vector R
         reflectanceMap[ texel++ ] = 255 * CookTorrance(specular.x, R, N);
         reflectanceMap[ texel++ ] = 255 * CookTorrance(specular.y, R, N);
         reflectanceMap[ texel++ ] = 255 * CookTorrance(specular.z, R, N);
      }
   }
}
```

Program 6.22: *Sampling Cook and Torrance model to compute a reflectance sphere map.*

a reflectance map can represent any arbitrary isotropic modulation function. The isotropic restriction comes from sphere-mapping.

Figure 6.23 shows a reflectance map for polished gold sampled using our technique for RGB=(0.63, 0.56, 0.37). Notice the low reflectance at normal incidence/reflection (center of the disk) and the high reflectance at grazing angles (in the outer ring).

6.7.3 Results for Mirror-like Reflections

Figure 6.24 shows results of using the reflectance sphere mapping technique on a pyramid made of copper. Notice the overall difference in reflection intensity on the three visible surfaces of the pyramid,

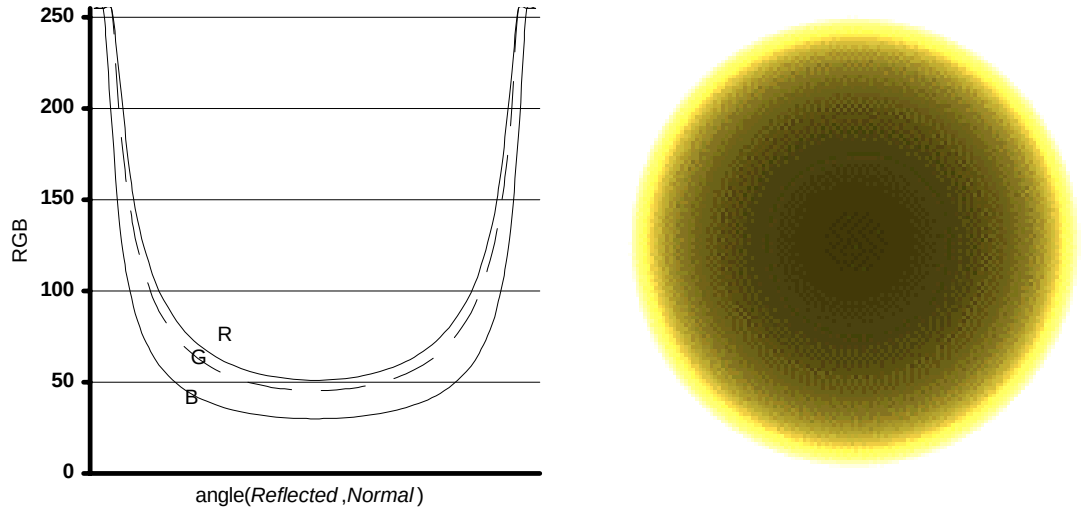


Figure 6.23: *Reflectance sphere map for gold obtained by sampling the Fresnel equation for RGB=(0.63, 0.56, 0.37). The graph shows RGB components along the horizontal line crossing the center of the sphere map. Note on both the texture and the graph the color shift predicted by the Fresnel equation.*

due to their distinct orientations with respect to the viewer. Also observe the color shift and reflection intensity of the specular reflection across the base of the pyramid.

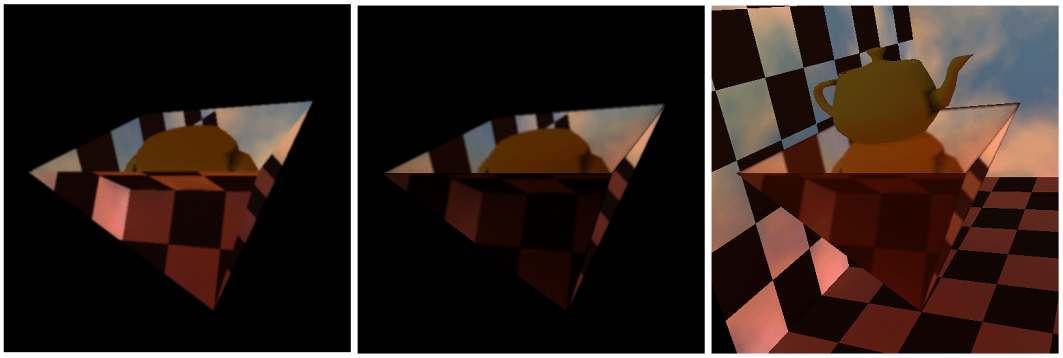


Figure 6.24: *Mirror-like reflection on three faces of a copper-like pyramid: (left) mirror reflection using geometry, (middle) mirror-like reflection obtained by modulating the mirror reflection from left image with the reflectance sphere mapping for copper, and (right) final image for scene reference.*

Figures 1.7 and 1.8 show results of using the directional reflectance sphere mapping for a glossy floor; the blurriness of the glossy reflection will be discussed in the next chapter, but the view-dependent reflectance was produced with the sphere mapping technique discussed in this chapter. In Figure 1.7 the viewing direction is parallel to the floor, whereas in Figure 1.8 the model was tilted such that the viewing direction is at a less grazing situation. Notice the difference in reflectance of the floor between the two images and the variation of the reflectance along the floor on both images.

The sphere mapping-based technique for approximating directional reflectance is very effective. Besides providing a good approximation of directional reflectance, sphere mapping is available in graphics hardware from SGI^{TM}, which makes it a fast operation. However, sphere mapping suffers from both inherent and implementation problems. The first inherent problem is that the mapping from viewing directions to texel coordinates in a sphere map is non-linear, which implies that directional resolution is not constant in the map. Secondly, as with any other texture mapping technique, each vertex of a triangle or quadrilateral (primitive) is mapped onto a texture by computing texture coordinates. The edges of a primitive are mapped to straight lines connecting adjacent vertices of the primitive in texture space. This mapping of edges of the primitive to straight lines in the texture map is valid for rectangular textures, but incorrect for sphere maps—primitive edges should map to arcs in sphere mapping. The first problem with current implementations is that connected vertices in a texture map are linearly interpolated for filling in the corresponding primitive. However, given the non-linearity of the mapping function, bilinear interpolation is not the appropriate interpolation for sphere mapping. A second implementation problem, related to the two previous problems, is that filtering is usually performed with a square filter in a rectangular texture. Sphere mapping should use a wedge-shaped filter defined in terms of the radius and theta coordinates of sphere mapping, instead. Finally, current implementations of sphere mapping in the hardware do not perform the expected wrapping of the sphere map texture. Consider a triangle seen from a grazing angle. Then, the three vertices map to the outer region of a sphere map, and the filled triangle should get texels only from the outer (grazing) region of the sphere map. Suppose that the three vertices have the same θ value and that one of the vertices has a ϕ value and the other two vertices have the opposite ϕ value plus and minus some small angle. The correct mapping when filling in the corresponding triangular region should wrap the sphere map to provide texels only from grazing regions. However, the current implementation crosses the sphere map and incorrectly provides texels that go through the non-grazing region.

Our particular use of the sphere mapping technique for directional reflectance stresses the non-linearity problem of the technique. Our application requires high resolution on a region (outer ring) of sphere-maps where the sphere-mapping technique offers low resolution. Though we reduced this problem by uniformly increasing the resolution of the whole map, a better solution would be to flip the sphere-mapping indexing scheme. Grazing angles would index texels in the center of the sphere-map, where there is higher resolution than in the ring.

6.8 Discussion

Traditionally in computer graphics non-diffuse reflections are approximated with lighting models—the illumination of a point in a scene takes into account only the direct light from primary light sources. This is a reasonable approximation for surfaces on which highlights are more prominent than scene reflection (plastics). However, all non-diffuse materials exhibit reflections of surrounding environments to some extent. Instead of a view-dependent lighting approach for producing highlights based only on light sources, a reflection approach and an associated view-dependent reflectance scheme are necessary. This chapter discussed the rendering of mirror reflections and its extension to mirror-like reflections (when the materials exhibit directionally dependent reflectance).

The first part of this chapter addressed the rendering of non-scattered light transfers. The complexity of rendering mirror reflections with polygonal scenes was recognized and an image-based approach presented to provide better performance. We should now realize that in image-based rendering there are two different quantities to be reconstructed:

1. The geometry of the scene needs to be reconstructed so that we can generate images from novel view poses. Since images-with-depth preserve the three-dimensional location of each pixel in the scene, the pixels can be reprojected to generate new images.

2. The illumination of the original scene also needs to be reconstructed; we want to preserve view-independent details such as shadows and view-independent light distribution in the scene. Since images-with-depth preserve color information per pixel, by reprojecting the points and by controlling their reprojected size, we reconstruct both illumination and geometry for novel viewpoints.

However, note that after the geometry of the original scene has been sampled, there is no distinction between geometry and illumination information anymore. Illumination information is now directly coupled to geometrical points. This somehow goes against the observations of Chapter 5, where illumination information was decoupled from geometry information by transforming radiosity information into textures. For example, a 512×512 image-with-depth of a single quadrilateral now contains 256K point samples. So, instead of geometrically transforming four points of the quadrilateral and filling the projected region on the screen, the image-based approach requires transforming 256K points and filling the same region on the screen. There is a need for converting the point samples back

to the original continuous planar primitive in order to avoid unnecessary geometrical transformations and sampling and reconstruction issues.

As an alternative to images-with-depth, we could have used images-with-id.s. Instead of storing the color and depth at each pixel of an image, we would store an identifier (id.) of the corresponding primitive visible at that pixel. This would allow assembling a list of all the primitives visible for each image, i.e., the potentially visible set of primitives (PVS). A reprojection of such an image-based entity would be done by just sending the PVS to the pipeline for the current view pose. Since this alternative process would not involve any resampling, it would benefit directly from the primitive rendering and the reconstruction schemes already available in the hardware. Moreover, the amount of data to be rendered for each PVS would potentially be much smaller than the amount of data to be rendered using the corresponding image-with-depth.

The image-based approach also suffers from disocclusion artifacts. A single image-with-depth captures the first visible point from the COP in the direction of each pixel. When an image-with-depth is reprojected to a novel COP, unsampled regions are exposed. Our approach minimized disocclusion artifacts by reprojecting multiple images taken from different COPs around the desired COP, for producing the final result. The distinct COPs around the desired COP increased the sampling of occluded regions of a single image-with-depth. However, because the selected COPs were in a small neighborhood, the corresponding images captured similar information—a same surface could be captured by all the images-with-depth. Since a surface could be represented in multiple images-with-depth, reprojecting a set of images-with-depth for reconstructing a new desired view implies in some redundancy. A pixel in the desired view may be touched several times due to the multiple images-with-depth sampling the same point in the scene. A more efficient approach would be based on layered-depth-images (LDIs) [Gortler97]. An LDI works in the same way as an image-with-depth, but captures not only the first visible point at each pixel, but also further points along the direction from the COP to each pixel. A standard way of creating an LDI uses a set of images-with-depth reprojected to the LDI view pose. Each pixel in the LDI contains an array of points for storing multiple layers of depth from the LDI viewpoint. Pixels from different images-with-depth that reproject into the same three-dimensional location or into a small 3D neighborhood are stored as a single entry in the pixel array for the average location. A single LDI captures more information about the scene and produces less disocclusion artifacts when reprojected to novel COPs than a single image-with-depth. The points-reduction scheme of LDIs reduces the redundancy factor from the set

of images-with-depth to the LDI. Consequently, fewer LDIs are necessary to capture similar visibility information as a given number of images-with-depth.

Although simple to implement, the splatting-like zero-order reconstruction scheme used in this chapter for images-with-depth leads to noticeable artifacts in the reflected images. A more convolution-oriented reconstruction scheme is needed. A better reconstruction scheme would be based on real splatting of Gaussian kernels associated to the point samples.

The second part of the chapter discussed mirror-like reflections or how to extend mirror reflections to simulate reflections of materials that have directionally dependent reflectance. Although important for rendering a wide variety of materials, the direction-dependent reflectance is usually not taken into account when computing reflections. This chapter discussed a two-pass approach where the directionally dependent reflectance was introduced with a rendering pass over the mirror-like surface in conjunction with a view-dependent texture-mapping technique—sphere mapping—available in current graphics hardware APIs such as OpenGL.

Previous approaches for rendering the ideal mirror-reflection on a planar surface were based on the techniques discussed in Section 6.3, which imposed doubling the complexity of the scene to be rendered [Diefenbach96, McReynolds98]. To make reflections from real silvered-glass mirrors look right, the light must be attenuated at least by 8% for the two air-glass surfaces encountered and perhaps tinted to capture the greenish transmissivity of the glass. For these and other mirror-like reflections, the most common approach is to modulate the mirror-reflected image with a single view-independent color [McReynolds98]. That approach is able to approximate only colored mirrors. Diefenbach [Diefenbach96] discussed the use of hardware fog features to produce view-dependent shading of the mirror-reflected scene based on the distance of the reflected points to the mirror plane. That approach provides a linear approximation where points close to the mirror surface get maximum intensity, whereas points farther from the mirror get smaller intensities. Note, however, that although this approach is view-dependent, the approximation does not incorporate any physical property of the surface material (BRDF).

CHAPTER 7

SCATTERED TRANSFERS OF LIGHT

So far we have discussed the two extremes of the light scattering range. Chapter 5 addressed the ideally diffuse (fully scattered) transfers, which distribute light from any single incoming direction equally in all outgoing directions. Chapter 6 considered the ideally specular (non-scattered) transfers of light, which distribute light in a single outgoing direction for incoming light in any single direction. In nature, however, most materials are neither perfectly diffuse nor perfectly specular. Real surfaces scatter light into solid angles due to the microscopic roughness of the surface (Figure 7.1)[1]. This chapter discusses the in-between or non-ideal transfers—glossy transfers—which perform a certain level of scattering in particular directions, instead of the equally distributed scattering of ideally diffuse transfers or the non-scattered behavior of ideally specular transfers. The unequal directional scattering implies that glossy transfers are view-dependent.

The radiosity method (Chapter 5) greatly simplified the light transport problem for ideally diffuse transfers by precomputing a view-independent quantity—radiosity—for real time rendering. The light transport equation was also greatly simplified for ideally specular transfers in Chapter 6, where the hemispherical integration was reduced to considering incoming light along a single direction of the whole incoming hemisphere. Glossy transfers range from almost ideally diffuse to almost ideally specular and neither simplification applies. Outgoing light in a single direction from a glossy transfer at a small neighborhood around point $\mathbf{x}$ is the result of scattering light from a non-zero incoming solid angle due to microscopic surface roughness (Figure 7.1). The solid angle can range from almost the

[1] Note that Figure 7.1 and the text describing light transfers above implicitly evoke the reciprocity principle of light transfers (Section 3.5.5). The text above follows the conventional description of light transfers and BRDFs as the scattering of a single incoming ray of light into an outgoing solid angle with distribution given by the BRDF at the transfer point (Figure 3.5). The reciprocity principle states that if the direction of propagation of light is reversed, the light path remains unchanged. That is, reversing the direction of propagation of light in the text above implies that the scattering of an incoming solid angle of light produces light into a single outgoing direction (Figure 7.1).

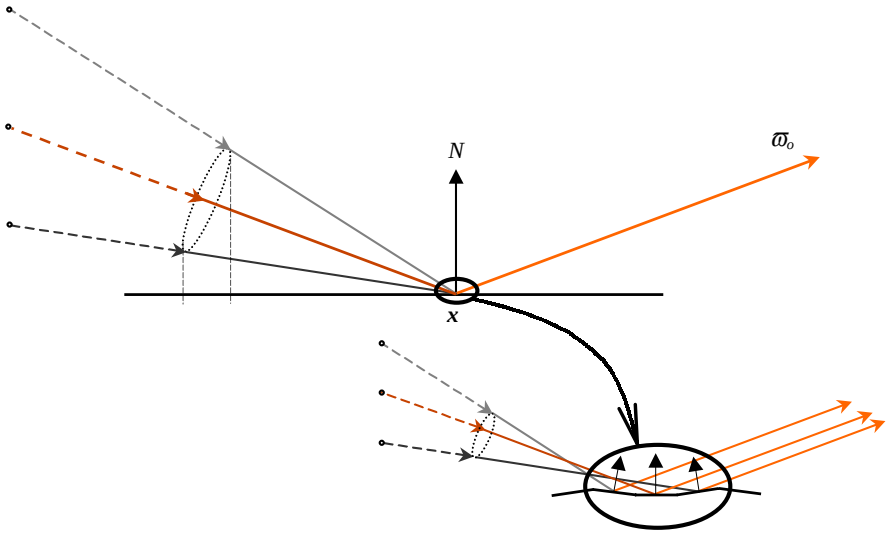


Figure 7.1: *Microscopic roughness of the surface at a neighborhood around point* x *produces light in a single outgoing direction* $\vec{\omega}_o$ *due to a glossy transfer.*

entire hemisphere down to almost a single direction, depending on how close the glossy material is to an ideally diffuse material or to a mirror, respectively. The BRDF lobe bounds the solid angle ω for which the material responds to incoming light for transferring light into a single outgoing direction $\vec{\omega}_o$ (Figure 7.2). The BRDF lobe is centered around the incoming direction $\vec{\omega}_{io}$, which derives from the desired outgoing direction $\vec{\omega}_o$ and represents the incoming direction of maximum contribution to the given outgoing direction. The principal direction $\vec{\omega}_{io}$ is usually the mirror-reflected direction of the desired outgoing direction $\vec{\omega}_o$. An exception is due to the increasing reflectance at grazing angles, where possibly the mirror-reflected direction may not represent the direction of maximum contribution from a grazing incoming solid angle. However, the shift in the direction of maximum incoming contribution from the mirror-reflected direction does not represent an approximation, since the BRDF lobe captures the correct distribution function in the neighborhood of the incoming solid angle.

This chapter presents methods for rendering glossy reflections based on the convolution approach to light transport (Chapter 4). Translucency, or glossy transmission, is left as an extension of the techniques presented. Section 7.2 reviews the convolution approach to glossy transfers. Then, Section 7.3 and Section 7.4 discuss object-space convolution and image-space convolution methods for approximating glossy reflections.

Each of the object-space and the image-space methods is split into two distinct parts based on the nature of light transfers. The first part deals with the visibility from the point where the light transfer takes place—the incoming light or *what* is visible from the glossy surface. The second part deals with

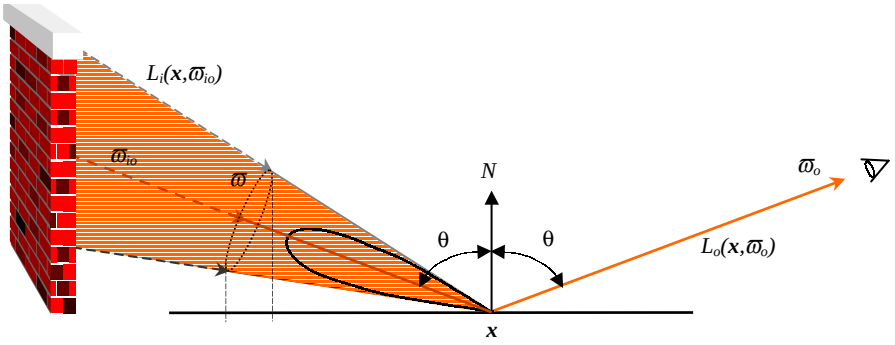


Figure 7.2: *Incoming radiance integration for a small solid angle subtending a BRDF lobe around direction* $\vec{\omega_{io}}$.

material properties at the point in question—*how* the material responds to incoming light for producing transferred light at the outgoing (viewing) direction.

The object-space method performs the visibility part with a particle-tracing phase by shooting light particles from the light sources and by storing the incoming direction of the particles at each hit of a particle with a surface in the scene. The image-space methods perform the visibility part by computing the mirror-reflected image on glossy surfaces and by assuming pixel-to-pixel visibility coherence in the reflected image. The visibility part in the object-space approach is a preprocessing stage, whereas in the image-space approaches it is a runtime phase of the rendering process. In practice, the visibility preprocessing in the object-space method constrains its application to low gloss reflections. Near-mirror reflections would require preprocessing incredible amounts of directional information (incredible numbers of particle hits in the scene) in the object-space method. The image-space method, instead, is more applicable for high gloss reflections, since it is based on mirror-reflected images.

Given the visibility information for the glossy surface, both methods use convolution to perform the material properties part of the task. Since the material properties of a glossy material are view-dependent, the convolution process involved in the material properties part is expected to be fully view-dependent. The object-space method performs view-dependent convolution by associating a convolution kernel (splat) with each particle-hit point in object-space and by view-dependently modulating the magnitude of each splat based on the view-dependent BRDF at the particle hit. The superposition of the screen projection of all the object-space splats represents the convolution of the incoming light with the material properties at each particle-hit point on the glossy surface. Since particle-hit splats are defined in object-space, projecting all the splats on the screen and summing all the contributions does not produce a correct result. Only the splats belonging to visible surfaces have to be considered for the convolution. A depth test is necessary to select the splats to be considered. All the

splats are projected on the screen, but only those that land on visible surfaces have their contributions added to the final result. This selection process implies that many splats (belonging to occluded surfaces) are handled without producing any effect in the final result. The object-space method has a computational cost that is linear in the scene complexity.

The image-space methods perform convolution in screen-space using two different approaches. The first approach splits the material properties into two steps: a view-dependent modulation and a view-independent (spatially invariant) convolution. The modulation step view-dependently modulates the mirror-reflected image computed in the visibility part, according to the view-dependent magnitude of the BRDF lobe. Then, for approximating solid angle integration, a convolution step performs spatially invariant (view-independent) convolution of the image produced by the view-dependent modulation step. This two-step approach assumes that the shape of the BRDF lobe does not change view-dependently. A Phong lobe ($\cos^n \theta$), for example, has the shape-invariant property. The magnitude of the BRDF lobe, however, is allowed to vary view-dependently to produce higher specular contributions at grazing angle than at normal incidence, for example. The combination of the view-dependent modulation and the view-independent convolution represents a view-dependent process that is able to capture only part of the character of glossy transfers. The spatially invariant convolution implies two limitations of the approach. First, spatially invariant convolution cannot represent the fact that the shape of BRDF lobes is in fact view-dependent. Secondly, since the spatially invariant convolution is performed in projected (screen) space, the convolution kernel support in projected space is related to the neighborhood that approximates visibility in per-pixel reflection solid angles. The kernel support is approximately proportional to the distance from the reflection point to the reflected point af each pixel in the mirror-reflected image. However, since spatially invariant convolution assumes a constant kernel support across the domain (reflected image) the neighborhood size is constant per-pixel, which implies that all pixels would have the same depth value—an inconsistency. Our second convolution-based approach to the material properties step overcomes these two limitations.

The second approach that we present for the material properties step takes into account the full character of glossy material properties by performing spatially variant convolution of the mirror-reflected image with per-pixel material properties kernels. Since the convolution kernel is allowed to vary from pixel-to-pixel, the on-screen spatially variant convolution is view-dependent and the resulting method is capable of capturing the complete material properties across the glossy surface.

Additionally, the screen-space spatially variant convolution also allows for the appropriate per-pixel kernel support selection to approximate reflection solid angle visibility per-pixel.

The next sections discuss how we split the transferred term of the light transport equation into two parts, and then treat object-space and image-space convolution-based approaches to glossy reflections.

7.1 Splitting Glossy Light Transfers into Two Parts

The finite solid angle implied by glossy BRDF lobes reduces the integration range in the light transport Equation (4.1). Integration for glossy transfers can be done over a solid angle ω bound by the BRDF glossy lobe, instead of over the whole hemisphere Ω:

$$L_o(\mathbf{x}, \vec{\omega_o}) = L_e(\mathbf{x}, \vec{\omega_o}) + \int_{\omega} f_{bd}(\mathbf{x}, \vec{\omega_o}, \vec{\omega_i})\ L_i(\mathbf{x}, \vec{\omega_i})\ cos\ \theta_i\ d\vec{\omega_i} \tag{7.1}$$

where terminology is equivalent to Equation (4.1).

Chapter 4 split the light transport equation into an emitted term and a transferred term. Here we tailor the transferred term for glossy transfers:

$$transferred(\mathbf{x}, \vec{\omega_o}) = \int_{\omega} f_{bd}(\mathbf{x}, \vec{\omega_o}, \vec{\omega_i})\ L_i(\mathbf{x}, \vec{\omega_i})\ cos\ \theta_i\ d\vec{\omega_i} \tag{7.2}$$

where $transferred$ is the outgoing light resulting from the glossy transfer of light at point $\mathbf{x}$, and L_i is the incoming light. The glossy scattering at point $\mathbf{x}$ is due to microscopic roughness at the point and it is taken into account with the BRDF f_{bd}, which weights the incoming light at each of the incoming directions in the solid angle ω.

We split the transfer of light at any point $\mathbf{x}$ into two distinct parts: the light visible from the point in the incoming glossy solid angle—the *what* is visible from the point—and the material properties governing the scattering of light into the direction of the viewer—the *how* light is transferred by the point. The *what* is visible from the point is represented by the incoming radiance L_i in the transferred light Equation (7.2). The *how* light is transferred is represented by the BRDF f_{bd} at point $\mathbf{x}$ in Equation (7.2).

This splitting of a light transfer into two distinct parts indicates a general method for computing light transfers as a two-step approach. Both the object-space and the image-space methods that we present in the next sections exploit this splitting of glossy transfers into distinct visibility and material steps.

7.2 Convolution Approach to Glossy Transfers

Glossy transfers are the ones that most benefit from a convolution-based approach to the light transport problem. From Section 4.1, we know that we can write the light transport equation in terms of convolution:

$$L_o(\mathbf{x}, \vec{\omega_o}) = L_e(\mathbf{x}, \vec{\omega_o}) + F(\mathbf{x}, \vec{\omega_i}) \star K_s(\mathbf{x}, \vec{\omega_{io}}, \vec{\omega_i}) \tag{7.3}$$

where $F(\mathbf{x}, \vec{\omega_i})$ represents the incoming radiance to point $\mathbf{x}$ in direction $\vec{\omega_i}$ and $K_s(\mathbf{x}, \vec{\omega_{io}}, \vec{\omega_i})$ is the convolution kernel, which represents a re-parameterization of the specular (glossy) BRDF properties at point $\mathbf{x}$. That is, the transferred term in the light transport equation can be seen as the response of a system with input signal F and impulse response K_s. Note also how F and K_s relate directly to the two parts of a light transfer defined in the previous section. The input signal F represents *what* is visible from point $\mathbf{x}$, and the convolution kernel K_s represents *how* the system responds to an impulse input signal.

Expressing the radiance equation in terms of spatially varying convolution in hemispherical space is an interesting mathematical tool, but it does not yield a performance speed up *per se* for computing the radiance leaving the surfaces of a scene. Current hardware implements convolution in image-space, and we need another step to let us perform radiance convolution in that space and to benefit from the hardware.

The next sections describe an object-space convolution approach [Stuerzlinger97] and an image-space convolution approach [Bastos99] for approximating Equation (7.3) at interactive rates.

7.3 Object-space Convolution Approach to Light Transport

Our object-space approach for approximating the light transport equation is split into two phases. The first phase—*particle-tracing*—computes the distribution of light in a scene by tracing particles through the environment with a stochastic shooting approach. Each light source in the scene emits a number of light particles, which are each traced through the scene until being absorbed or leaving the scene. Each hit of a particle with surfaces of the scene is recorded for use in the next phase. The second phase—*runtime rendering*—reconstructs images using the hit records computed in the first phase with a feed-forward convolution approach. The contribution of each particle hit is scaled view-dependently by the BRDF of the surface and it is spread over a small neighborhood around the hit point. The superposition

of all the contributions reconstructs the view-dependent light distribution leaving the surfaces of the scene.

7.3.1 Particle-Tracing Phase

The particle tracing phase simulates how light interacts with a scene by tracing light particles through the environment in a stochastic shooting manner [Shirley95, Walter97b]. Each light source i in a scene emits a number of light particles proportional to the total power Φ_i of the light source. The direction of each particle depends on the emission distribution function associated with its light source—diffuse light sources emit according to a uniform distribution, whereas directional light sources emit in a range of directions.

Each particle is traced through the scene from its origin at the light source until it is absorbed or it leaves the scene. A random number chooses between absorption and reflection whenever a particle hits a surface. Absorbed particles are no longer propagated, whereas reflected particles follow an outgoing direction stochastically selected according to the BRDF of the material. All the particle hits in the particle path are stored, along with the incoming direction of the particle at each hit, corresponding light source, the current number of bounces of the particle, and the accumulated attenuation of the particle until it hits the current surface. Even though the quantity needed for our computations is the particle power ϕ, by storing the attenuation factor ρ_ϕ, instead of the particle power, we enable the method to change the number of particles emitted by a light source without affecting previously emitted particles from the same source. The power of a particle at a hit can then be computed from the power of the light source Φ_i divided by the number of particles m_i emitted by that source and weighted by the current attenuation factor of the particle hit:

$$\phi = \frac{\Phi_i}{m_i}\rho_\phi . \tag{7.4}$$

This scheme allows for incremental refinement of the solution, as the number of light particles does not need to be fixed *a priori*. The information stored for each particle hit includes the corresponding surface, the position (u, v) on the surface, the incoming direction, the light source where the particle originated, the number of bounces (transfers) of the particle so far, and the total attenuation factor due to previous transfers.

Figure 7.3 shows a visualization of particle hits density on the surfaces of a simple scene with one million particle hits.

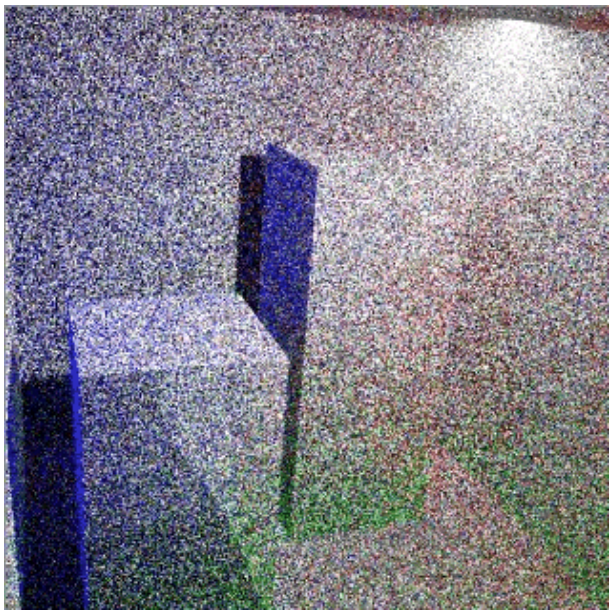


Figure 7.3: *Visualization of particle hit density.*

7.3.2 Runtime Rendering Phase

The directionally dependent radiance $L_o(\mathbf{x}, \vec{\omega_o})$ visible at a viewing direction $\vec{\omega_o}$ from a surface point $\mathbf{x}$ is reconstructed using the particle hits stored in the first phase. The contribution of each particle ϕ_j with incoming direction $\vec{\omega_{i_j}}$ is scaled by the BRDF of the surface at the point $\mathbf{x}_j$ to take into account the characteristics of the surface. To achieve a smooth reconstruction of the outgoing radiance over the surface, the contribution of each particle is spread over a small neighborhood, weighted by a kernel function k centered at each particle hit location $\mathbf{x}_j$:

$$L_o(\mathbf{x}, \vec{\omega_o}) = \frac{1}{h^2} \sum_{j=1}^{n} f_r(\mathbf{x}_j, \vec{\omega_{i_j}}, \vec{\omega_{o_j}}) \phi_j \; k(\frac{\mathbf{x} - \mathbf{x}_j}{h}) \tag{7.5}$$

where k is the kernel function (a normalized Gaussian kernel) and h is a spatial scaling factor that defines the kernel support. The size of the kernel support is adjusted for each surface in the scene and depends on the density of particle hits on each surface:

$$h = C\sqrt{\frac{A}{n}} \tag{7.6}$$

where A is the surface area, n is the total number of particles incident on the surface, and C is a scaling factor. Equation (7.6) controls reconstruction sharpness based on the number of particles impinging on a surface and on the surface area. The radiance reconstruction on a surface improves with increasing number of particles incident upon that surface. Increasing the number of incident particles, n, increases

the visibility information about the scene as seen by the surface and captures more accurate radiance information. Accordingly, increasing number of incident particles decreases the kernel support h to spread the contribution around each hit point.

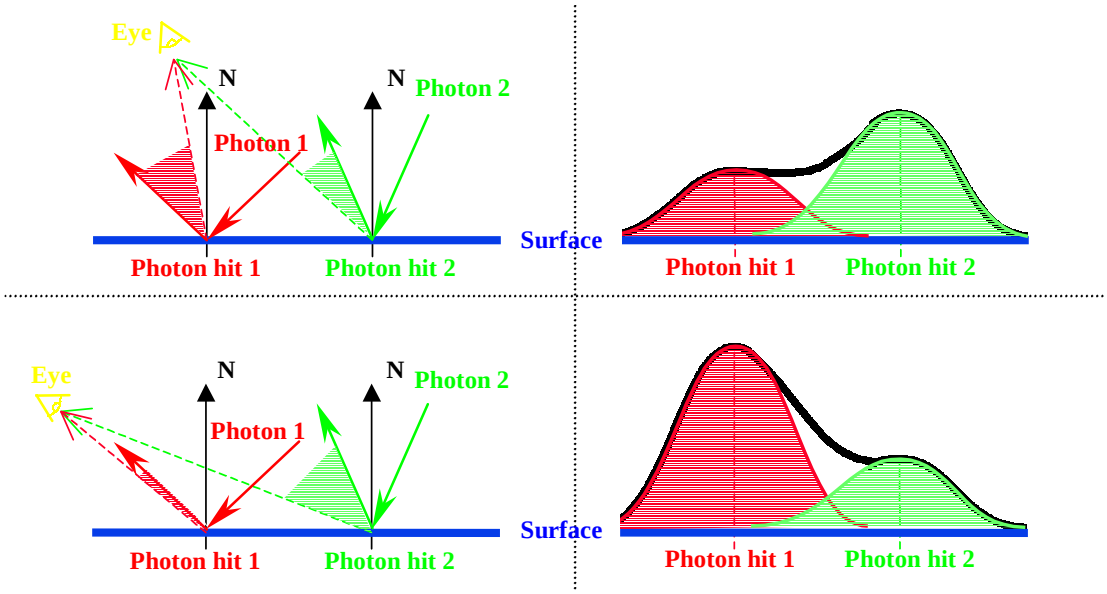


Figure 7.4: *Particle tracing (left) and splatting (right) for two viewing situations (top and bottom) on a horizontal surface: the particle tracing involves only two light particles (photon 1 and photon 2). Notice that the incoming directions and the corresponding ideally reflected directions for the particle hits do not change with the viewer. However, the BRDF for each particle hit varies view-dependently. In terms of splatting, each hit point is associated with a Gaussian splat (right). The magnitude of the splat is given by the corresponding BRDF value modulating a normalized kernel. The superposition of the splats on the right gives the view-dependently-reconstructed outgoing radiance from the surface.*

A key observation is that Equation (7.5) amounts to splatting the contribution of each particle onto its respective surface (Section 2.6.2.2). That is, the equation represents the convolution of the incoming light with a density-based and BRDF-based convolution kernel. Each hit point on a surface is associated with a Gaussian kernel, and the superposition of all the kernels on the surface reconstructs the outgoing radiance for a given view pose (Figure 7.4). All quantities but the BRDF in Equation (7.5) are constant for each surface. The BRDF is a view-dependent quantity and has to be evaluated for each particle at each frame. The evaluated BRDF at each hit point controls the magnitude of the kernel at that point (Figure 7.4).

Reconstruction is performed with splatting. The contribution of each particle hit is taken into account by rendering a triangle on the plane of the corresponding surface. The triangle is constructed to enclose the support of the kernel centered at the hit point (Figure 7.5). The color of the triangle is

determined per frame by multiplying the power of the particle at the hit (Equation (7.4)) by the BRDF of the material under the current viewing conditions and the corresponding incoming direction of the particle (a table look up). The kernel texture mapped onto the triangle modulates the triangle color.

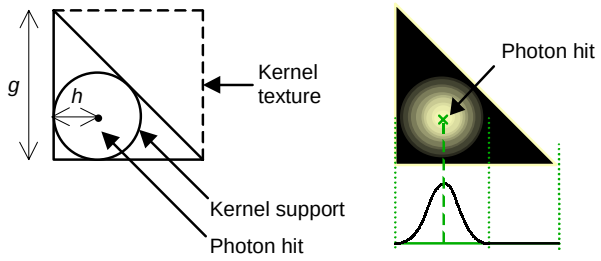


Figure 7.5: *Geometric relationships for the triangle and the kernel support for a particle hit. The triangle normal vector is aligned with the surface normal at the hit point.*

In terms of implementation, splatting is performed by rendering a textured triangle for each particle hit (Figure 7.5) and adding the corresponding pixel values to the color buffer. The texture is a sampled Gaussian function. The geometry of the required triangles for representing the particle hits is precomputed. The kernel texture for splats is also precomputed. A triangle is created for each particle hit, but a single kernel texture is used for all the splats in the scene. The texture is mapped onto each particle-hit triangle. The reconstruction algorithm proceeds as presented in Program 7.6, for a preprocessing step, and in Program 7.7, for a runtime step.

```
Compute the kernel texture
For each surface
   Determine the size of the kernel splat for the surface
   For each particle hit recorded for the surface
      Create a triangle on the plane of the surface and size it to enclose the kernel support.
      Map the kernel texture onto the triangle.
```

Program 7.6: *Preprocessing step.*

The splatting is performed using graphics hardware blending functions set up to sum rendered pixel values to the color buffer. The kernel function is precomputed as a texture with 64×64 texels. Mip-mapping is used for texture minification (filtering). Figure 7.5 illustrates the geometric relationships of the triangle, the kernel support, and the kernel texture. In the figure, h is the scaling factor defined in Equation (7.6) and $g = (1/2 + \sqrt{2})h$.

```
Render the scene into the depth-buffer
For each surface
   Render the surface to the stencil buffer
      Set stencil at pixels where surface depth equals depth-buffer value
   For each particle hit on this surface
      Evaluate the BRDF based on the surface material, for the particle incoming direction,
         and for the viewer outgoing direction
      Set the color of the triangle to the power of the particle hit multiplied by the BRDF
         of that particle hit and viewer location.
      Add the splat to the color buffer (by rendering the triangle with additive blending)
```

Program 7.7: *Runtime step.*

7.3.2.1 Optimized rendering phase

Direct illumination accounts for the most prominent illumination features: highlights and shadows [Bastos96]. Factoring out the direct illumination from the global illumination significantly reduces the number of photons that need to be rendered. The direct illumination of a point light source onto each surface is rendered using simulated Phong-shading [Diefenbach96] and shadow maps [McReynolds98]. Current graphics hardware systems support only per-vertex Phong lighting for point light sources. To overcome this limitation, each surface is subdivided into a fine regular mesh during setup. This mesh is displayed using per-vertex graphics hardware Phong lighting and bilinear interpolation. Small-enough area light sources can be simulated with point light sources without introducing significant error. Direct illumination from larger area light sources can be simulated by multiple light sources.

Shadow mapping is used to account for occlusion of the light source. The original polygons of the scene are rendered into a depth texture from the viewpoint of the light source. This depth texture is used during rendering to identify shadowed parts of the scene. Multiple light sources require multiple shadow maps.

Indirect lighting is taken into account by summing contributions of particles that have been reflected more than once. The modified algorithms are shown in Program 7.8 and 7.9.

7.3.3 Results for the Object-Space Approach

The object-space method presented in the previous sections was implemented in C++ using the OpenGL library. Results were obtained on an SGI Onyx workstation using a single 250MHz

```
Compute the kernel texture
For each surface
   Determine the size of the kernel splat for the surface
   For each particle hit recorded for the surface
      Create a triangle on the plane of the surface and size it to enclose the kernel support.
      Map the kernel texture onto the triangle.
Create a shadow map for each light source.
Create a fine triangular mesh for each surface by regular subdivision.
```

Program 7.8: *Preprocessing step.*

```
Render the scene into the depth-buffer.
For each light source
   Activate the corresponding shadow map.
   Add the direct illumination to the color buffer by rendering all surfaces
      as triangular meshes with Phong lighting and shadow maps.
For each surface
   Render the surface to the stencil buffer
      Set stencil at pixels where surface depth equals depth-buffer value
   For each particle hitting this surface that has already been reflected by another surface
      Evaluate the BRDF based on the surface material, on the particle incoming direction, and
         on the viewer outgoing direction
      Set the color of the triangle to the power of the particle hit multiplied by the BRDF
         of that particle hit and viewer location.
      Add the splat to the color buffer (by rendering the triangle with additive blending)
```

Program 7.9: *Runtime step.*

R4400 processor and an InfiniteReality graphics pipe with two raster managers and 12 bits per color component in the color buffer.

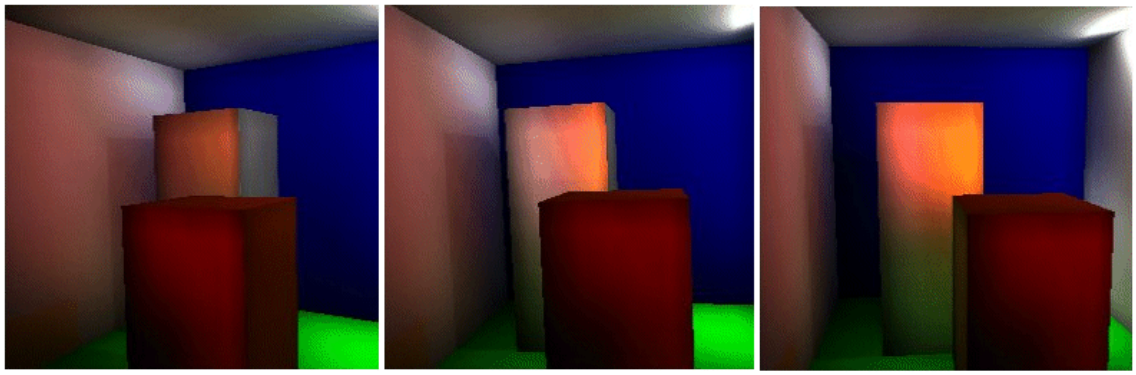


Figure 7.10: *Image sequence showing how the glossy reflection on the large box changes with viewing angle. The red wall behind the viewer changes the color of the reflection on the glossy box.*

Figure 7.10 shows results of the method for a simple scene with a glossy box (the taller box). Particle tracing in this scene generated approximately one million particle hits (Figure 7.3) which are rendered at an average of 0.25 frames per second on the machine described above and at 1.2 frames per second with the optimized rendering of Section 7.3.2.1 on the same machine. The same scene runs at 5 fps on a more recent workstation (SGI Onyx2 using a single 300MHz R12000 processor and an InfiniteReality2 graphics pipe with four raster managers).

The main factor determining performance of the method is the time to draw the textured triangles, i.e., rendering the primitives and computing the feed-forward convolution. Since smooth reconstruction requires extensive overlap among the splats, the total rendering time is dominated by the time to fill the projected triangles on the screen. Each pixel on a neighborhood around the projected hit point on the screen receives contributions (sums to the color buffer) from several splats (textured triangles) that overlap at that neighborhood.

7.3.4 Discussion of the Object-Space Approach

The object-space approach presented so far approximates the light transport equation in image-space by accumulating contributions of projected kernels defined in object-space. Each particle hit is associated with a kernel-textured triangle coplanar to the corresponding surface in object-space. The accumulated projection of the textured triangles reconstructs the outgoing radiance of scattered transfers on glossy surfaces. The reconstruction process involves handling (rendering and accumulating) a scene with a very large number of kernel-textured triangles corresponding to particle hits. The approach

involves a convolution of the outgoing radiance from each particle hit, which requires rendering many kernel-textured triangles with extensive overlap, besides rendering the original geometry of the scene. The significant overlap of triangles in a neighborhood produces a similar effect to increasing depth complexity of the scene (Section 7.3.2.1). Several triangles are rendered to each pixel to produce a single pixel value. Even though depth complexity can be reduced with occlusion culling methods [Zhang97] when rendering opaque geometry, the pixel complexity produced by convolution cannot be avoided. All the kernel contributions mapping to a pixel need to be taken into account to compute the final convolution value at that pixel. Even with ideal culling algorithms for opaque geometry, the convolution approach has to sum all the radiance kernel contributions mapping to each pixel. If occlusion culling is not available, then even occluded kernel-textured triangles need to be rasterized, but only the triangles belonging to visible geometry need to have their values summed to the color buffer.

The object-space convolution approach has a visibility assumption. It assumes that no visibility events happen on the neighborhood (kernel support) around each particle hit point. That is, it assumes that the scene point where a light particle originates is visible for any point across the particle hit neighborhood. This assumption is violated at shadow regions when the kernel support should be chopped at the shadow boundary to avoid light leaking across the boundary between the lighted region and the shadow region. A shadow here means occlusion between the point where a particle originates and the particle hit point, but the particle origin is not necessarily a primary light source. The perceptual result of not handling shadow discontinuities is blurrier shadow boundaries. Since glossy surfaces produce blurry reflections anyway, this effect is not noticeable.

The object-space approach uses the convolution kernel partially as a view-independent reconstruction entity and partially as a view-dependent reconstruction entity. To provide smooth reconstruction of the outgoing radiance, the approach estimates the kernel support for each particle hit based on particle hit density on the corresponding surface—in a view-independent manner. To provide view-dependent outgoing radiance, BRDF information is used to scale the magnitude of each kernel-textured triangle. Note that convolution kernels in the object-space approach depend on the BRDF information only for the magnitude of the kernels. The kernel support and the kernel function do not depend on material properties.

The object-space method has a computational cost that is linear in the number of particle hit points in the scene.

7.4 An Image-Space Convolution Approach to Light Transport

A simpler and more efficient alternative to the object-space approach is to use image-space convolution to approximate the light transport equation. The image-space approach, like the object-space approach, decouples visibility information from material properties information in light transfers, but it follows one of two alternative two-step techniques for producing glossy reflections. First, visibility information is computed with a mirror reflection of the scene on the glossy surface. Then, the material properties of the glossy surface are taken into account by processing the mirror-reflected image to approximate how the material scatters and transfers light in the direction of the viewer.

The image-space approach has a visibility assumption similar to that of the object-space approach. Visibility events in the pixel neighborhood around each pixel of a glossy surface are disregarded by assuming that the mirror-reflected image on the glossy surface neighborhood captures the same visibility information as the reflected solid angle at the pixel in question. Unlike the object-space approach, the image-space approach allows using the convolution kernel as a complete material properties entity, instead of a purely smoothing entity. Moreover, the image-space approach can achieve better computational time by performing convolution for at most the number of pixels on the image, rather than for the multiple superimposed particle-hit splats of the object-space approach.

The next sections detail the visibility step and the material properties step of our image-space approaches. The visibility step discusses how visibility inside the reflected solid angle at each pixel of a glossy surface can be approximated with a pixel neighborhood around the pixel in question on the mirror-reflected image of the glossy surface. The material properties step uses the above visibility approximation to weight and integrate incoming light over reflected solid angles at each pixel of the glossy surface. The view-dependent weighting and integration is performed with image-space convolution with two different techniques. The first technique splits the desired view-dependent convolution process into two steps: a view-dependent directional modulation based on the magnitude variation of the glossy BRDF and a spatially invariant (view-independent) convolution in screen space. The second technique uses spatially variant (view-dependent) convolution in screen space. The spatially variant convolution determines the convolution kernel for each pixel of a glossy surface based on the depth of the mirror-reflected point and the BRDF at that same pixel.

7.4.1 Visibility Step: Coherence in Mirror-Reflected Images

Light transfers can be split into two independent parts (Section 7.1): a visibility part and a material properties part. The two parts are defined in the same domain, i.e., in the directional space around the point where the light transfer takes place. In theory, the domain is the entire hemisphere above the point in question—a table of hemispherical BRDF data is needed for representing the material properties at the point and a hemispherical image is needed for capturing the hemispherical visibility at the same point on a reflective surface. In practice, for a glossy transfer, both parts have to be considered only over a finite solid angle bound by the glossy BRDF lobe at the point in question. A window of the hemispherical BRDF data is enough and an image through the corresponding reflected solid angle at each pixel is enough. The visibility step in this section discusses how we can approximate visibility for reflected finite solid angles by assuming pixel-to-pixel visibility coherence in mirror-reflected images.

The visibility step of our two-step approach starts by computing the mirror-reflected image of the scene on each given glossy surface. We restrict our analysis to planar glossy surfaces and we use the techniques from Chapter 6 for computing mirror-reflected images on planar surfaces. Note that ray tracing could also be used for computing the mirror-reflected images, since our two-step approximation depends only on the final rendered mirror image and not on the particular technique used for computing the reflection. Both color and depth information at each pixel of the mirror-reflected image are necessary for approximating reflected solid angle visibility for glossy reflections.

Our visibility assumption says that a mirror-reflected image captures most of the visibility information needed for computing glossy reflections. Figure 7.11 illustrates how visibility for a glossy reflection solid angle at a point on a horizontal surface can be approximated with mirror reflection on a neighborhood around that point. The visible region for solid angle ω with apex at point $\mathbf{x}$ can be approximated by the visibility of mirror-reflected rays for neighboring points around $\mathbf{x}$. The scene point visible for the top ray of the solid angle in the figure is captured by the mirror-reflected ray at point $\mathbf{x}_l$ on the surface. Similarly, the visibility of the bottom ray of the solid angle is captured by the mirror-reflected ray at $\mathbf{x}_r$. In general, the visibility for any ray inside the solid angle is captured by the mirror-reflected ray at a point in the neighborhood between $\mathbf{x}_l$ and $\mathbf{x}_r$. This suggests that a mirror-reflected image captures enough visibility information for approximating solid angle visibility at each point of a glossy surface.

In terms of implementation, let's consider what happens in the depth and color buffers when approximating visibility for glossy solid angles at pixels of a glossy surface. A glossy planar surface

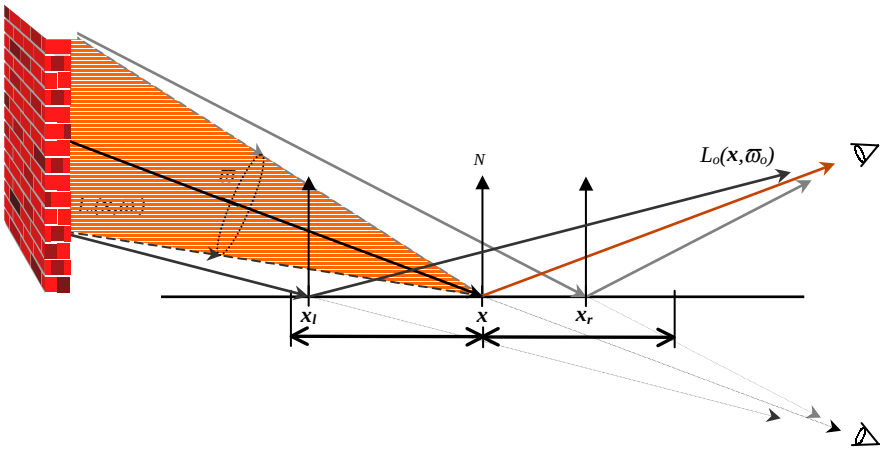


Figure 7.11: *Exploiting coherence in a mirror-reflected image for approximating solid angle visibility at a pixel* $\mathbf{x}$*. The visibility for the solid angle centered at* $\mathbf{x}$ *is approximated by the visibility for the reflected rays at the neighborhood around point* $\mathbf{x}$ *and between* $\mathbf{x}_l$ *and* $\mathbf{x}_r$*. The solid angle top ray visibility is captured by the reflected ray at* $\mathbf{x}_l$*, while the visibility for the bottom ray of the solid angle is captured by the reflected ray at* $\mathbf{x}_r$*.*

visible for a given view pose covers a region of the color and depth buffers. Using the techniques discussed in Chapter 6, the corresponding region in the depth and the color buffers contains the depth (distance from the reflection point on the surface to the reflected point in the scene) and the color of the mirror-reflected scene point at each pixel.

Note that coherence on the mirror-reflected image on a surface produces only an approximation of visibility for non-zero reflection solid angles (glossy reflections). Since the visibility from the point in question and the visibility from another point in the neighborhood capture different viewpoints of the scene, visibility events (occlusions and disocclusions) can happen from one point to another. That is, the visibility of the mirror-reflected scene on the neighborhood around a point on a surface can produce results that differ from the desired non-zero solid angle visibility for the same point. Our techniques for glossy surfaces calculate as if this does not happen. Precisely when reflections are glossy (somewhat blurred) rather than ideally specular, the artifacts arising from violations of this assumption are rarely noticeable.

7.4.2 Material Properties Step: Image-Space Convolution for Integration Over Solid-Angles

The second step of our two-step approach to light transfers deals with how light over an incoming glossy solid angle is weighted and integrated (convolved) with the material BRDF to produce light into a single

outgoing (viewing) direction. Theoretically, this convolution takes place in the hemispherical space above a point (differential area) on a glossy surface (Chapter 4 and Section 7.2). Our technique works in image-space by approximating weighted solid angle integration using image-space convolution.

Image-space convolution performs a gathering of contributions around each pixel on the mirror-reflected image, which approximates the solid angle integration required for evaluating the light transport equation at each pixel. The gathering neighborhood is defined by the convolution kernel support, which derives from the same pixel neighborhood used for approximating solid angle visibility. The convolution kernel is a texture in the same space as the reflected image—a sampled version of the kernel function, which represents the weighting value for each incoming and outgoing directions pair. In terms of the convolution operation, the kernel can be seen as a sliding window that is shifted across the input domain (image). To evaluate the convolution at a pixel $\mathbf{x}$, the kernel is centered at that location in the image and the sum of the pixel-wise products of the two images (reflected image and convolution kernel) is taken. The next two sections discuss our alternative material properties steps that use spatially invariant and spatially variant convolution

As was the case in the previous section, the convolution of mirror-reflected images to approximate reflected solid angle integration implies a visibility assumption. Each scene point ideally-reflected by a glossy surface is assumed to be visible for all the points on the neighborhood defined by the kernel support around the point in question. This visibility approximation is similar to the one described for the object-space approach.

7.4.2.1 Material properties step using spatially invariant convolution and directional modulation

The convolution in the light transport equation is spatially variant in hemispherical space. The BRDF kernel changes not only its magnitude but also it shape in directional space. Moreover, in image space, convolution is also spatially variant due to perspective shortening on the mirror-reflected images used to capture visibility information. In this section we disregard the second part of the spatially variant character of the problem and discuss an approach that performs a view-dependent (spatially variant) modulation of the mirror-reflected image and then performs a spatially invariant (view-independent) convolution of the resulting image. The motivation for this approach is the possibility of exploiting operators available in the current graphics hardware—view-dependent texture mapping and spatially invariant image-space convolution.

The view-dependent modulation step is performed according to Section 6.7.1 using sphere mapping. The directional modulation step then needs to model only the magnitude changes of the BRDF lobe in directional space, not the shape change. The spatially invariant convolution models the glossy integration around the small neighborhood at each pixel of the mirror-reflected and modulated image. The convolution kernel is an image representing the shape of the invariant kernel.

Program 7.12 presents pseudo-code for feed-forward spatially invariant convolution in image-space—an input image produces an output image with the same resolution convolved with a kernel image. In Program 7.12, the convolution kernel is spatially invariant over viewing directions (pixels).

```
zero all pixels in the output image
for each pixel q in the input image
   center kernel in the output image at the same location q
   for each pixel p in the output image covered by the kernel
      output value at p += input value at q * kernel value at p
```

Program 7.12: *Feed-forward image convolution using spatially invariant kernel.*

7.4.2.2 Material properties step using spatially variant convolution

Program 7.12 in the previous section assumed a spatially invariant kernel, i.e., that the base shape and weighting values of the kernel do not change from pixel to pixel. However, the convolution kernel used for approximating a glossy reflected image may have to change across a glossy surface due to the view dependence of both the magnitude and the shape of the BRDF. Spatially invariant convolution of a mirror-reflected image performs neighborhood gathering of pixel information based on a constant (invariant) kernel support across the reflected image. The kernel support, shape, and weighting function depend on the material properties of the reflective surface—the kernel represents the material BRDF. However, in our image-space approximation, the kernel also depends on another variable—the distance from each reflection point on the glossy surface to the reflected point on the scene.

Let's consider the evaluation of light transport at two points on the glossy floor of a simple scene (Figures 7.13 and 7.14). First, consider the mirror-reflected direction for the point x on the floor. The reflected ray in Figure 7.13 hits the far brick wall, whereas the reflected ray in Figure 7.14 hits a closer point. Clearly, the distance d from the floor point (reflection point) to the point hit on the scene (reflected point) is different for the two situations. Now consider the solid angle visibility approximation at

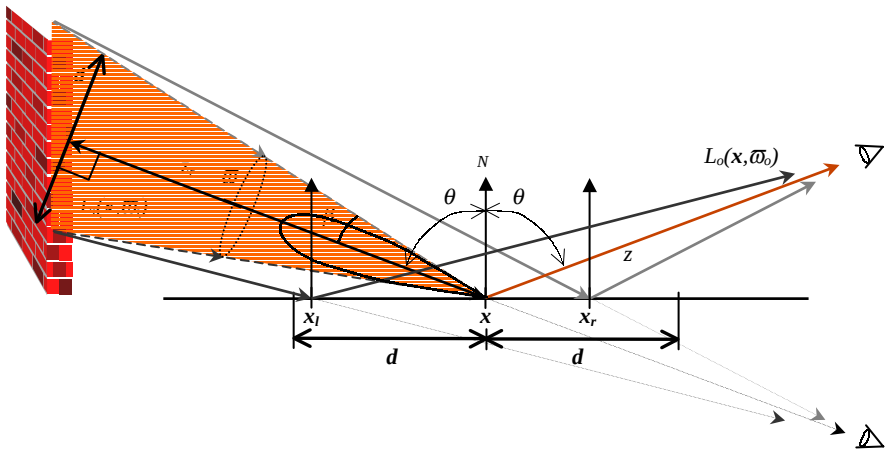


Figure 7.13: *Estimating the neighborhood size (kernel support) for point* x *on the glossy surface for a far reflected point on the brick wall.*

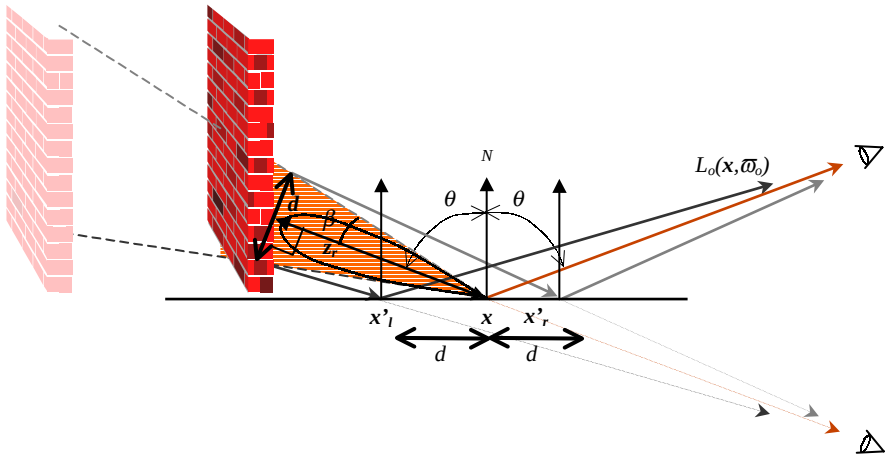


Figure 7.14: *Estimating the neighborhood size (kernel support) for point* x *on the glossy surface for a nearer reflected point.*

point $\mathbf{x}$ for both situations. Since the point $\mathbf{x}$ is exactly at the same for both situations, the BRDF lobe has the same shape for the two situations and the subtended solid angle is also the same for both situations. However, the neighborhoods on the reflected image (on the floor) that capture visibility around the floor point have different sizes. Under our visibility approximation, the neighborhood size $2d$ depends on the distance z_r from the reflection point on the floor to the reflected point in the scene (Figure 7.14). We estimate the kernel support for each pixel on a reflected image with the following formula for the kernel diameter in image space:

$$k_r = 2d = 2(2z_r \tan \beta) \tag{7.7}$$

where d is the diameter of an imaginary disk defined by the intersection of the solid angle and a plane orthogonal to the principal direction $\vec{\omega_{io}}$ at the reflected point, z_r is the distance from the reflection point to the reflected point (the depth value in the reflected image), and β is the half angle around the principal direction $\vec{\omega_{io}}$ defining the solid angle. Clearly, for the two situations in Figures 7.13 and 7.14, $\tan \beta$ is constant but z_r has different values, which implies distinct kernel supports for the two situations.

Note that the desired behavior is captured by the kernel support of Equation (7.7). The support increases for increasing solid angles—the bigger the BRDF lobe, the wider the visibility neighborhood to be considered and the larger the kernel support. The kernel support is also directly proportional to the distance from the reflection point to the reflected point—the further the reflected scene from the glossy surface, the blurrier the glossy reflected image. The kernel support also vanishes when the solid angle is infinitesimal or when the distance from the reflected point to the reflection point vanishes—the reflection degenerates to a mirror-like reflection.

Getting convolution kernels from BRDFs

The spatially varying image-space convolution requires two parameters for the kernel at each pixel: the kernel support and the kernel texture. These parameters are determined using depth information and BRDF information at each pixel. The kernel support defines the size of the pixel neighborhood to consider for the solid angle visibility approximation. At each pixel, the kernel support is a function of the solid angle subtending the BRDF lobe, and the distance of the reflected point with respect to the reflection point (Equation (7.7)). The BRDF lobe at each pixel defines the angle β in Equation (7.7)—at runtime, this is a look-up into a table of $\tan \beta$ indexed by the $\cos \theta$ at the pixel. The kernel texture represents the magnitude of the BRDF for each direction in the incoming solid angle and for the single outgoing viewing direction. Imagine a kernel texture as a curved patch on a sphere

centered at the glossy transfer point (Figure 7.15). The patch covers the solid angle on the sphere for

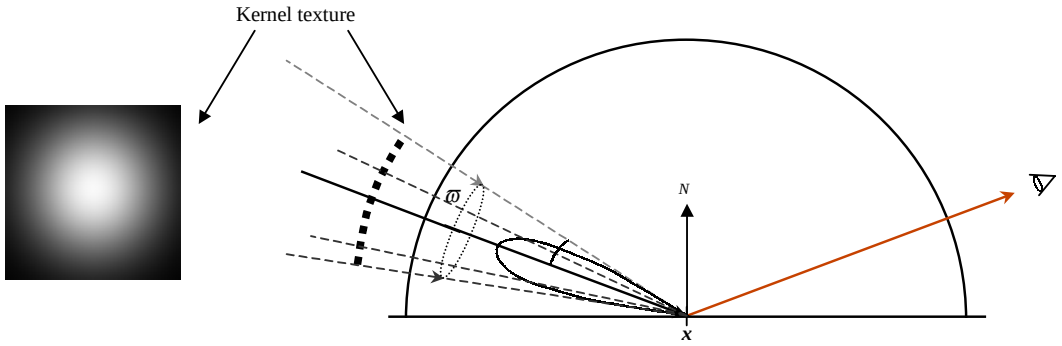


Figure 7.15: *A kernel texture (left) seen as a patch on a hemisphere (right) covering the region for which a BRDF at point* x *(in the center of the hemisphere) has non-zero values. For a given outgoing direction, each texel samples the BRDF at a direction in the incoming non-zero solid angle.*

which the BRDF at the point and the outgoing direction in question has non-zero values. Each texel samples the BRDF in the corresponding incoming direction on the sphere. A library of kernel textures is created for each directionally dependent BRDF function. At runtime, the kernel texture is selected from the library of kernels based on the outgoing direction with respect to the surface normal ($\cos\theta$) at each pixel.

7.4.3 A Feed-Forward Implementation

The image-space convolution described in the previous section requires a spatially variant kernel in image space, which currently is not available in graphics hardware. For proof of concept, we have implemented a software version of spatially variant feed-forward convolution (splatting), as presented in Program 7.17. We hope that in the future graphics hardware will implement spatially variant convolution, which will make our convolution-based technique approach real time for rendering globally illuminated scenes with arbitrary BRDFs.

Our technique takes a mirror-reflected image and a material library as inputs and produces an approximation of the glossy reflected image on the corresponding surface with the given material. The input image is the region in image-space covered by the projection of the glossy surface. The material library is a set of kernel textures and a table of directional kernel supports sampled as described in the previous section. Figure 7.16 illustrates the spatially variant splatting, and Program 7.17 describes the algorithm for our image operation. In terms of an OpenGL implementation, the technique assumes that a mirror-reflected image has been computed for the glossy surface and the corresponding pixel color

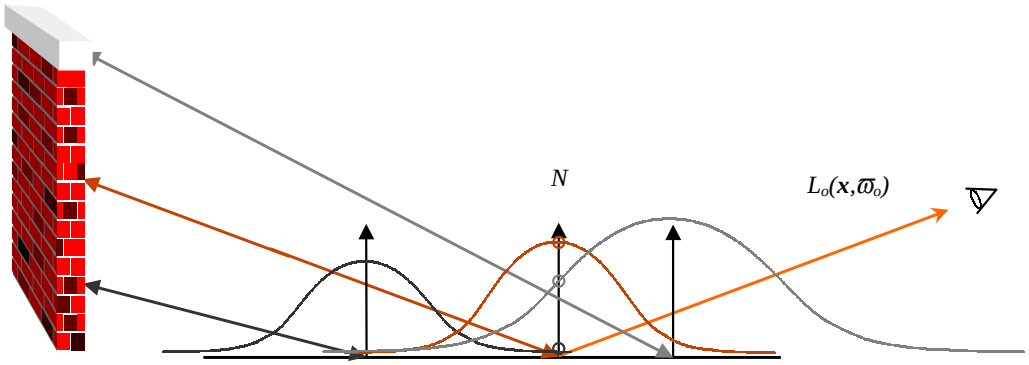


Figure 7.16: *Spatially variant splatting: three splats are shown. Notice the different support (width) and magnitude (height) of each splat.*

and depth values are in the color and depth buffers. The technique then proceeds by reading into main memory the color buffer and the depth buffer regions containing the mirror-reflected image to generate a new image of the same resolution, which approximates the evaluation of the transferred term in light transport equation at each pixel. The convolution kernel is estimated for each pixel in the input image based on the depth of the reflected point, the BRDF information, and the surface normal at each pixel.

```
zero all pixels in the output image
for each pixel q in the input image
   look up the BRDF lobe size based on outgoing direction and surface normal at pixel q
   estimate kernel support based on BRDF lobe size and depth of reflected point at pixel q
   select kernel texture based on outgoing direction and surface normal at pixel q
   center kernel in the output image at the same location q
   for each pixel p in the output image covered by the kernel
      output value at p += input value at q * kernel value at p
```

`Program 7.17:` *Image-space spatially varying splatting for approximating the light transport equation at pixels of a glossy surface.*

A hardware implementation of our technique could be similar to SGI's implementation of image-space spatially invariant convolution [McReynolds98]. The operation would be performed in a feed-forward manner by transferring an image from a region in memory into another region in memory (for example, from the frame buffer into main memory or into texture memory). The initial or source image would be the unconvolved image and the resulting or destination image would be the convolved image. The convolution would be done by splatting the RGB value of each original pixel into the neighborhood around that same pixel in the destination image, according to a convolution

kernel. That is, the RGB value of each original pixel would modulate a kernel texture and the weighted result would be summed to the destination image. In the spatially invariant implementation, the convolution kernel is the same across the whole image, whereas for spatially variant convolution the kernel would be allowed to change for each pixel in the image. For our technique, a library of mip-mapped kernels would be pre-loaded into texture memory and a different kernel would be selected for each original pixel. The per-pixel selection would be based on the per-pixel quantities needed by the desired type of convolution. For example, for directionally invariant but spatially variant kernels, which represent some types of BRDFs in glossy reflections, the selection process would reduce to evaluating Equation 7.7 for each pixel. Note that in the directionally invariant case, $\tan \beta$ is a constant, and evaluating Equation 7.7 requires multiplying the depth value at each pixel (from the depth buffer) by a constant.

7.4.4 Results for the Image-Space Approach

The results presented in this section were obtained on an SGI Onyx2 workstation using a single 300MHz R12000 processor and an InfiniteReality2 graphics pipe with four raster managers.

Figure 7.18 shows a comparison of spatially invariant (left) and spatially variant (right) image-space convolution for approximating a glossy reflection on a planar surface. Notice the distance-based variable blur obtained with spatially variant convolution (right); this is not produced with the spatially invariant convolution (left). For example, points of the teapot are very close to the glossy surface and do not get blurred much with spatially varying convolution, whereas points along an edge of the checkerboard pattern on the wall get progressively blurrier with increasing distance to the glossy surface. This variable blur provides an important depth cue about the three-dimensionality of the scene with respect to the glossy surface as it is expected in real glossy reflections. The spatially invariant image-space convolution (left) benefits from a hardware implementation from SGI^{TM} and renders in times on the order of tens of milliseconds (tens of frames per second). The spatially variant image-space convolution was implemented in software and renders in times on the order of minutes. A more careful software implementation of spatially variant convolution could provide better performance, but certainly the greatest performance improvement will come from a hardware implementation of spatially variant convolution.

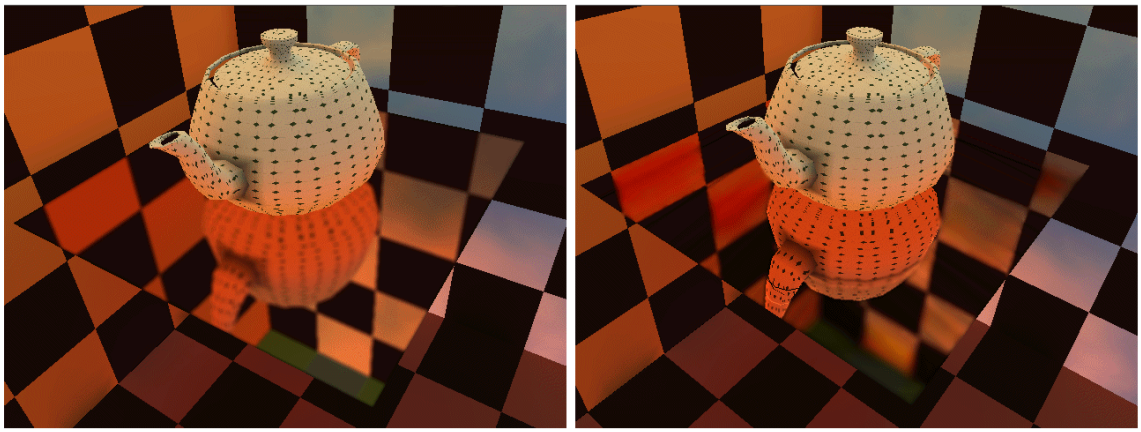


Figure 7.18: *Spatially invariant convolution (left) and spatially variant convolution (right) to approximate glossy reflection on planar surface.*

7.5 Discussion

Traditionally in computer graphics, scattered light transfers are grossly approximated with lighting models that compute the illumination at a point by taking into account light reaching the point only directly from light sources. In nature, however, we can easily observe that a glossy reflection involves not only direct light from the sources, but also indirect light from the entire environment visible to the reflective surface. Reflection models are necessary, instead of direct lighting models.

Efforts have been made to approximate glossy reflections by the superposition of manipulated lighting models [Walter97a]. Colored point light sources were created around glossy objects to approximate the incoming light from the surrounding environment onto the glossy surface. The summation (superposition) of the lighting resulting from all those point light sources approximates the glossy reflection of the environment. This technique is effective for low gloss surfaces when a small number of light sources is enough to roughly approximate the incoming light from the environment onto the glossy surface. However, how many light sources would be necessary to represent the incoming light onto a high gloss surface? In a high gloss reflection it is possible to distinguish reflected objects from the environment. This approach is impractical for high gloss surfaces—it would require too many point light sources to approximate the environment around high gloss surfaces.

As an alternative, multipass rendering can approximate scattered light transfers with a stochastic approach [Diefenbach96]. The entire scene is rendered multiple times for jittered reflected viewpoints with respect to the glossy surface. The weighted average (superposition) of all the images approximates the glossy reflection of the environment on the glossy surface. This approach is effective when just a few jittered re-renderings of the scene are enough, i.e., for high gloss surfaces. However, how many

jittered re-renderings would be necessary for approximating a low gloss reflection? In a low glossy reflection reflected objects get very blurred and hard to distinguish. This approach is impractical for low gloss surfaces—it would require too many scene re-renderings of the entire scene for approximating such type of glossy reflections.

Note that the two approaches described above are implicitly computing a convolution (superposition). The incoming light onto the glossy surface is the input signal, the material properties of the glossy surface define the convolution kernel, and the outgoing glossy reflection is the output signal. This chapter exploited explicit convolution approaches for approximating glossy reflections. Both object-space and image-space convolution approaches were presented. The image-space approach processes smaller amounts of data (number of pixels on the screen) than the object-space approach (complexity of the scene). In fact, the object-space approach depends not only on the complexity of the scene, but also on the number of light particles colliding with surfaces of the scene (a much larger number than scene complexity). Both approaches were discussed in terms of flat surfaces, but they extend to arbitrary geometrical primitives under the visibility assumption discussed in the chapter. The visibility assumption disregards visibility events (occlusion and disocclusions) in small neighborhoods around points where light transfers take place. Violations of the visibility assumption may create blurred regions where sharper edges would be expected.

The spatially variant image-space convolution presented in this chapter operates in image-space and seems simple enough for a graphics hardware implementation. In contrast to other approaches for computing glossy reflections, the presented convolution approach does not require handling excessive information about the scene at each pixel of the final image. Global information of the scene at each pixel of a glossy surface is extracted from a small neighborhood around a mirror-reflected image computed for the surface in question. Data locality in image-space is good for gathering the global information about the scene necessary for computing scattered light transfers.

The same spatially variant convolution algorithm discussed in this chapter for rendering glossy reflections can be used for rendering other effects such as translucency, soft-shadows, motion blur, depth-of-focus, and anti-aliasing. For the algorithm to be applicable, the desired effect must be expressed as the weighted average of global information in small neighborhoods in image space. The difference in how the convolution algorithm is used by the different effects is in the convolution kernel that needs to be used for each of the effects.

An important point about the object-space approach is that it solves the entire light transport problem at once, without handling independent components of light transport separately. This does not fit well with our multipass superposition rendering where we want to use different approaches to render the independent components, benefiting from the particular strengths of each approach. A possibility for using the object-space approach to render only glossy reflections in conjunction with a multipass technique would require isolating and rendering only the particle hits landing on glossy surfaces, after completion of the particle-tracing phase.

The object-space approach used the kernel mostly as a smoothing entity—taking into account the density of samples, whereas the image-space approach interpreted the convolution kernel as the entity describing the material properties.

Since the weighted contributions in the convolution approach can get very small, precision is an important issue. We have observed that eight bits per color component is usually not enough for evaluating convolution. Twelve bits per color component, in general, produced satisfactory results, but a full floating point word per color component for each pixel would certainly avoid generation and propagation of error when computing the convolution.

CHAPTER 8

FUTURE WORK

This dissertation opens interesting research directions for future work, which will be briefly sketched here in approximately the order the parent topics arise in the dissertation.

8.1 Mirror-Reflected Images

8.1.1 Reflected Images on Curved Surfaces

The mirror reflection approach used in this dissertation is limited to planar surfaces, but the extension to curved surfaces is possible. The planar surface reflection approach could be applied to each of the small planar facets of a tesselated curved surface. To produce a non-faceted reflection, an additional warping of the reflected points would be necessary for approximating the curvature of the faceted curved patches.

Although the method described above would be able to approximate geometrically correct reflections for curved surfaces, the use of reflection mapping (environment mapping) is more attractive due to its simplicity of implementation and low computational cost [Greene86]. Among other limitations, environment mapping assumes that the reflected environment is infinitely far way from the reflective surface, and it is not able to capture geometrically correct reflections [Bastos98]. However, reflections on curved surfaces are so visually complicated that it is hard to judge whether a reflection is geometrically correct or not.

8.1.2 Recursive Reflections

This dissertation limited its treatment to a single specular reflection along a path of light. This limitation is reasonable for glossy surfaces, since the brightness of the reflected environment is greatly reduced

and blurred at each glossy reflection and higher-order reflections are hardly noticeable. However, for more polished surfaces, the higher-order reflections may be necessary.

The rendering of higher-order reflections could be implemented with a recursion of the reflections technique presented. For a given view pose, the recursive reflections tree should be traversed and then reflected images rendered for each node of the tree in a bottom-up manner. For example, consider a scene with two glossy surfaces and a view pose that sees one of the glossy objects reflected on the other glossy object. The rendering process would first detect that there is a glossy object directly visible from the viewer. Then, from the corresponding reflected viewpoint on the object visible from the viewer, the rendering would detect the second glossy object. The recursion process would then stop, assuming no higher-order glossy reflections are wanted. Then, while popping up from the recursion, reflected images of the environment would be rendered at each node of the recursion, i.e., for each glossy object. In the final image, this would result on a second-order reflection composed on the first visible glossy object and a first-order reflection on the second glossy object.

8.2 Convolution-Based Approach

8.2.1 Convolution-Based Approach to Other Integral Equations

The convolution approach to the integral equation of light transport would certainly be applicable to other integral equations. Several physics problems are represented with integral equations, and their simulation usually involves solution techniques similar to the standard ones applied to light transport. The possibility of rewriting integral equations in terms of convolution and benefiting from convolution hardware to compute solutions of integral equations would probably accelerate the study of other interesting physics problems.

Although one normally thinks of the convolution hardware in current systems as image-processing hardware, we can alternatively understand that particular part of the hardware as a tool for solving numerical problems involving integration on small neighborhoods in two-dimensional spaces. The numerical problem has to be cast as the integration of the product of two functions in two-dimensional space. Then, one of the functions is selected the convolution kernel and the other function as the input signal, based on the nature of the problem. The kernel should be pre-computable and pre-loaded in memory, whereas the input signal should depend on each particular instantiation of the problem.

In addition to the convolution capabilities of the graphics hardware, the hardware-aided solution of numerical problems can also benefit from table look up operators (texture mapping) and from the linear operators addition and multiplication available in current graphics systems.

8.2.2 Full Digital Signal Processing Approach

This dissertation exploited only a few properties of signal processing applied to the light transport problem. Using the extensive knowledge from digital signal processing in the light transport problem would certainly present even more powerful alternative solutions. For example, a frequency analysis of the BRDF kernels in our light transport problem could be used to determine appropriate sampling rates for glossy materials with different reflectance properties.

8.2.3 Spatially Variant Convolution

8.2.3.1 Other Applications for the Spatially Variant Convolution

The spatially variant convolution approach to glossy reflections described in Chapter 7 can be used for other photorealistic effects, such as depth-of-field, motion blur, soft shadows, and translucency. For the spatially variant convolution algorithm to be applicable, the desired effect must be expressed as the weighted average of global information in small neighborhoods in image space. The difference in how the convolution approach is used by the different effects is in the convolution kernel required for each of the effects. For each pixel of an image to be produced with a given effect, a convolution kernel has to be selected according to the type of effect.

In glossy reflections, the convolution kernel shape in image space depends on the BRDF of the glossy material, whereas the kernel support at each pixel depends on the distance from the visible reflection point at a pixel to the corresponding reflected point in the scene. For depth-of-field, the kernel shape represents the impulse response of the camera lens, whereas the kernel support at each pixel is proportional to the distance from the visible point in a scene at each pixel to the in-focus plane of the lens. For motion blur, the kernel shape is related to the velocity vector of the visible point at each pixel with respect to the viewer, whereas the kernel support is proportional to the distance from the viewer to the visible point in the scene at each pixel. For soft shadows, the kernel shape represents the shape of the light source, whereas the kernel support depends on the distance from the light occluder to the visible point on the receiving surface at each pixel. For translucency, the kernel shape represents the impulse response function of the translucent material, whereas the kernel support depends on the

distance from the visible point on the translucent object at each pixel to the corresponding visible point in the environment seen through the translucent object.

8.2.3.2 Hardware Implementation of Spatially Variant Convolution

The software technique for spatially variant convolution presented in Chapter 7 is too slow for application in interactive rendering systems. However, the technique seems simple enough to justify considering a hardware implementation in current graphics architectures.

A possible implementation of the spatially variant convolution could be similar to SGI's implementation of image-space spatially **in**variant convolution [McReynolds98]. The operation would be performed in a feed-forward manner by transferring an image from a region in memory to another region in memory (for example, from frame buffer into main memory or texture memory). The initial or source image would be the unconvolved image and the resulting or destination image would be the convolved image. The convolution would be done by splatting the RGB values of each source pixel into the neighborhood around that same pixel in the destination image, according to the convolution kernel selected for the source pixel. That is, the RGB value of each source pixel would modulate a kernel texture and the weighted result would be summed to the destination image. In the spatially **in**variant implementation, the convolution kernel is the same across the entire image, whereas for spatially variant convolution the kernel would be allowed to change for each pixel in the image. For our technique, a library of mip-mapped kernels could be pre-loaded into texture memory and a different kernel would be selected for each source pixel. The per-pixel selection would be based on the per-pixel quantities needed by the desired type of convolution, as discussed in the previous section.

The availability of spatially variant convolution in hardware would certainly increase the realism of view-dependent effects in synthetic images that must be rendered at interactive rates.

8.3 Dynamic Environments

In Chapter 1, we limited this dissertation to the interactive rendering of static environments. The multipass approach presented in this dissertation depended upon precomputation of certain view-independent quantities for speeding up the rendering of static environments. In particular, the light paths ending in ideally diffuse surfaces were handled with the radiosity method. Given the high computational cost of solving the entire global radiosity problem of a scene, re-computation of radiosity per frame does not seem a possible solution for reasonably complex scenes. Efficient approaches for

updating radiosity information in dynamic environments could be a possible alternative. Approaches for computing reduced and more local radiosity solutions for each view pose could also be an alternative in dynamic environments.

BIBLIOGRAPHY

[Aliaga99] Daniel Aliaga, Jonathan Cohen, Andrew Wilson, Eric Baker, Hansong Zhang, Carl Erikson, Kenneth Hoff, Thomas Hudson, Wolfgang Stuerzlinger, Rui Bastos, Mary Whitton, Frederick Brooks, and Dinesh Manocha. Mmr: An integrated massive model rendering system using geometric and image-based acceleration. In *Proceedings of the 1999 Symposium on Interactive 3D Graphics*, 1999.

[Bastos93] Rui Bastos, António Augusto de Sousa, and Fernando Nunes Ferreira. Reconstruction of illumination functions using bicubic hermite interpolation. In *Rendering Techniques '93: Proceedings of the Eurographics Rendering Workshop 1993*, Paris, France, June 1993.

[Bastos96] Rui Bastos, Michael Goslin, and Hansong Zhang. Efficient rendering of radiosity using textures and bicubic reconstruction. Technical Report UNC-CH TR96-025, Univ. of North Carolina at Chapel Hill, Dept. of Computer Science, May 1996. Available at http://www.cs.unc.edu/Research/tech-reports.html.

[Bastos97] Rui Bastos, Michael Goslin, and Hansong Zhang. Efficient radiosity rendering using textures and bicubic reconstruction. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics*, Providence, RI, April 1997.

[Bastos98] Rui Bastos and Wolfgang Stuerzlinger. Forward mapped planar mirror reflections. Technical Report UNC-CH TR98-026, Univ. of North Carolina at Chapel Hill, Dept. of Computer Science, 1998. Available at http://www.cs.unc.edu/Research/tech-reports.html.

[Bastos99] Rui Bastos. Solving the radiance equation with convolution. Technical Report UNC-CH TR99-021, Univ. of North Carolina at Chapel Hill, Dept. of Computer Science, 1999. Available at http://www.cs.unc.edu/Research/tech-reports.html.

[Castleman96] Kenneth R. Castleman. *Digital Image Processing*. Prentice-Hall, 1996.

[Chen93] Shenchang Eric Chen and Lance Williams. View interpolation for image synthesis. In *Computer Graphics Annual Conference Series (Proceedings of SIGGRAPH 93)*, pages 279–288, Anaheim, California, August 1993.

[Cohen93] Michael F. Cohen and John R. Wallace. *Radiosity and Realistic Image Synthesis*. Academic Press, 1993.

[Collier71] Robert J. Collier, Christoph B. Burckhardt, and Lawrence H. Lin. *Optical Holography*. Academic Press, 1971.

[Cook81] Robert Cook and Kenneth Torrance. A reflectance model for computer graphics. *Computer Graphics (Proceedings of SIGGRAPH 81)*, pages 244–253, 1981.

[Diefenbach96] Paul Diefenbach. *Pipeline Rendering: Interaction and Realism Through Hardware-Based Multi-pass Rendering*. PhD thesis, University of Pennsylvania, 1996.

[Ditchburn91] R. W. Ditchburn. *Light*. Dover, 1991.

[Feynman89] Richard Phillips Feynman. *The Feynman Lectures on Physics*. Addison-Wesley, 1989.

[Foley90] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, second edition, 1990.

[Fowles89] Grant R. Fowles. *Introduction to Modern Optics*. Dover, second edition, 1989.

[Glassner94] Andrew Glassner. A model for fluorescence and phosphorescence. *Proceedings of Eurographics Workshop on Rendering 1994*, pages 57–68, 1994.

[Glassner95] Andrew Glassner. *Principles of Digital Image Synthesis*. Morgan Kaufmann, 1995.

[Gondek94] J. S. Gondek, G. W. Meyer, and J. G. Newman. Wavelength dependent reflectance functions. *Computer Graphics (Proceedings of SIGGRAPH 94)*, pages 213–220, 1994.

[Goral84] C. M. Goral, K. E. Torrance, D. P. Greenberg, and B. Battaile. Modeling the light between diffuse surfaces. *Computer Graphics (Proceedings of SIGGRAPH 84)*, pages 213–222, 1984.

[Gortler96] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. In *Computer Graphics Annual Conference Series (Proceedings of SIGGRAPH 96)*, pages 43–54, New Orleans, Louisiana, August 1996.

[Gortler97] Steven J. Gortler, Li wei He, and Michael F. Cohen. Rendering layered depth images. Technical Report #97-09, Microsoft Research, Mar 1997. Available at http://www.research.microsoft.com/pubs.

[Greene86] Ned Greene. Environment mapping and other applications of world projections. *IEEE Computer Graphics and Applications*, 6(11):21–29, November 1986.

[Greenler80] Robert Greenler. *Rainbows, halos, and glories*. Cambridge University Press, 1980.

[Heckbert90] Paul S. Heckbert. Adaptive radiosity textures for bidirectional ray tracing. *Computer Graphics (Proceedings of SIGGRAPH 90)*, pages 145–154, August 1990.

[Heckbert91] Paul S. Heckbert. *Simulating Global Illumination Using Adaptive Meshing*. PhD thesis, University of California, Berkeley, 1991.

[Heckbert92] P. Heckbert. Discontinuity meshing for radiosity. In *Rendering Techniques '92: Proceedings of the Eurographics Rendering Workshop 1992*, pages 181–192, Bristol, UK, May 1992.

[Hottel54] Hoyt C. Hottel. *Radiant Heat transmission*. McGraw Hill, New York, NY, 1954.

[Jensen96] Henrik Wann Jensen. Global illumination using photon maps. In *Rendering Techniques '96: Proceedings of the Eurographics Rendering Workshop 1996*, Porto, Portugal, June 1996.

[Kajiya86] James T. Kajiya. The rendering equation. *Computer Graphics (Proceedings of SIGGRAPH 86)*, 19(4):143–150, August 1986.

[Lafortune96] Eric P. Lafortune. *Mathematical Models and Monte Carlo Algorithms for Physically Based Rendering*. PhD thesis, Katholieke Universiteit Leuven, 1996. Available at http://www.graphics.cornell.edu/ eric/publications.html.

[Levoy96] Marc Levoy and Pat Hanrahan. Light field rendering. In *Computer Graphics Annual Conference Series (Proceedings of SIGGRAPH 96)*, pages 31–42, New Orleans, Louisiana, August 1996.

[Lischinski92] D. Lischinski, F. Tampieri, and D. Greenberg. Discontinuity meshing for accurate radiosity. *IEEE Computer Graphics and Applications*, 12(6):25–39, 1992.

[McMillan95] Leonard McMillan and Gary Bishop. Plenoptic modeling: An image-based rendering system. In *Computer Graphics Annual Conference Series (Proceedings of SIGGRAPH 95)*, pages 39–46, Los Angeles, CA, August 1995.

[McMillan97] Leonard McMillan. *An Image-Based Approach to Three-Dimensional Computer Graphics*. PhD thesis, University of North Carolina at Chapel Hill, 1997. Available as UNC-CH Computer Science TR97-013, at http://www.cs.unc.edu/Research/tech-reports.html.

[McReynolds98] T. McReynolds, D. Blythe, B. Grantham, and S. Nelson. *Advanced Graphics Programming Techniques Using OpenGL*. ACM SIGGRAPH, 1998.

[Mitchell88] Don Mitchell and Arun Nettravali. Reconstruction filters in computer graphics. In *Computer Graphics Annual Conference Series (Proceedings of SIGGRAPH 88)*, pages 221–228, August 1988.

[Moller96] Tomas Moller. Radiosity techniques for virtual reality - faster reconstruction and support for levels of detail. In *WSCG'96*, Plzen, Czech Republic, 1996.

[Myszkowski94] Karol Myszkowski and Tosiyasu L. Kunni. Texture mapping as an alternative for meshing during walkthrough animation. In *Rendering Techniques '94: Proceedings of the Eurographics Rendering Workshop 1994*, Darmstadt, Germany, 1994.

[Nishita85] T. Nishita and E. Nakamae. Continuous tone representation of three-dimensional objects taking into account shadows and interreflection. In *Computer Graphics Annual Conference Series (Proceedings of SIGGRAPH 85)*, pages 23–30, August 1985.

[Oppenheim96] Alan V. Oppenheim, Alan S. Willsky, and S. Hamid Nawab. *Signals and Systems*. Prentice-Hall, 1996.

[Parker98] Steven Parker, Peter Shirley, Yarden Livnat, Charles Hansen, and Peter-Pike Sloan. Interactive ray tracing for isosurface rendering. In *Visualization*, October 1998.

[Parker99] Steven Parker, William Martin, Peter-Pike Sloan, Peter Shirley, Brian Smits, and Chuck Hansen. Interactive ray tracing. In *Interactive 3D*, April 1999.

[Ratliff72] F. Ratliff. Contours and contrast. *Scientific American*, 226(6):91–101, 1972.

[Shirley95] Peter Shirley, B. Wade, P. M. Hubbard, D. Zareski, B. Walter, and D. P. Greenberg. Global illumination via density-estimation. In *Rendering Techniques '95: Proceedings of the Eurographics Rendering Workshop 1995*, June 1995.

[Sillion94] François X. Sillion and Claude Puech. *Radiosity & Global Illumination*. Morgan Kaufmann, 1994.

[Stuerzlinger97] Wolfgang Stuerzlinger and Rui Bastos. Interactive rendering of globally illuminated scenes. In *Rendering Techniques '97: Proceedings of the Eurographics Rendering Workshop 1997*, June 1997.

[Veach98] Eric Veach. *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, Stanford University, 1998.

[Wallace87] John R. Wallace, Michael F. Cohen, and Donald Greenberg. A two-pass solution to the rendering equation: A synthesis of ray tracing and radiosity methods. *Computer Graphics (Proceedings of SIGGRAPH 87)*, 21(4):311–320, July 1987.

[Walter97a] Bruce Walter, Gun Alppay, Eric P. F. Lafortune, Sebastian Fernandez, and Donald P. Greenberg. Fitting virtual lights for non-diffuse walkthroughs. *Computer Graphics (Proceedings of SIGGRAPH 97)*, pages 45–48, August 1997.

[Walter97b] Bruce Walter, Philip M. Hubbard, Peter Shirley, and Donald P. Greenberg. Global illumination using local linear density estimation. *ACM Transactions on Graphics*, 16(3):217–259, July 1997.

[Ward94] Gregory J. Ward. The radiance lighting simulation and rendering system. *Computer Graphics (Proceedings of SIGGRAPH 94)*, July 1994.

[Watt92] Alan Watt and Mark Watt. *Advanced Animation and Rendering Techniques: Theory and Practice*. Addison-Wesley, 1992.

[Westover91] Lee Alan Westover. *SPLATTING: A Parallel, Feed-Forward Volume Rendering Algorithm*. PhD thesis, University of North Carolina at Chapel Hill, 1991.

[Wolberg92] George Wolberg. *Digital Image Warping*. IEEE Computer Society Press, Los Alamitos, California, 1992.

[Woo96] Mason Woo, Jackie Neider, and Tom Davis. *OpenGL Programming Guide*. Addison Wesley, 1996.

[Zhang97] Hansong Zhang, Dinesh Manocha, Tom Hudson, and Kenneth Hoff. Visibility culling using hierarchical occlusion map. *Computer Graphics (Proceedings of SIGGRAPH 97)*, 1997.

[Zimmons99] Paul Zimmons and Rui Bastos. Exploiting sphere mapping. Technical Report UNC-CH TR99-028, Univ. of North Carolina at Chapel Hill, Dept. of Computer Science, 1999. Available at http://www.cs.unc.edu/Research/tech-reports.html.