PHYSICALLY BASED LIGHTING CALCULATIONS FOR COMPUTER GRAPHICS

BY

PETER S. SHIRLEY

B.A., Reed College, 1985

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1991

Urbana, Illinois

ABSTRACT

Realistic image generation is presented in a theoretical formulation that builds from previous work on the rendering equation. Previous and new solution techniques for the global illumination are discussed in the context of this formulation.

The basic physics of reflection and light transport are used to derive the rendering equation. The problem of generating an image is phrased in terms of evaluating the Global Radiance Function, which consists of radiance values for all points and directions. This formulation of the rendering equation differs from previous formulations by rigorously accounting for transparent surfaces.

The physical rules governing reflection are used to make improvements in reflection models. In diffuse transmission it is shown that light is filtered to the same extent regardless of which side of the surface the light comes from. This eliminates one of the parameters from previous diffuse transmission models. The microscopic structure of polished surfaces was used to justify coupling the diffuse and specular coefficients according to the Fresnel Equations. The Fresnel Equations are commonly used to vary the reflectivity of metal and transparent dielectrics, but have not been used before to vary the reflectivity of the polish and underlying diffuse substrate.

Image-based solution methods are phrased as a lazy evaluation of the Global Radiance Function; evaluation takes place for visible points. Several constraints were outlined for what part of the image function should contribute to each pixel, and a separable, symmetric filter is developed that satisfies these constraints.

A stochastic shadow ray generation method is introduced that reduces the number of shadow rays needed for scenes with multiple light sources. The sampling distributions used for shadow rays and other dimensions of the integral are evaluated by introducing to computer graphics the notion of discrepancy from numerical integration theory. The use of discrepancy provided some insight not given by the signal processing theory traditionally used in computer graphics. As part of this discussion a new sampling scheme, N-rooks sampling, is introduced. N-rooks sampling is shown to be as efficient to generate as jittered sampling, while often outperforming Poisson disk sampling. It also can generate distributions for any positive integer number of samples, including primes.

The peculiarities of the sampling spaces used in distributed ray tracing are shown to preclude naive hierarchical sampling. It is demonstrated that hierarchical sampling can greatly reduce noise, however, if we have sufficient knowledge of the sampling space.

Zonal methods represent the opposite extreme of image methods, where all function values are computed and stored, and each evaluation is a table lookup. The zonal method is phrased as a transport simulation, similar to progressive refinement radiosity methods. Using this direct simulation model, it is straightforward to generate zonal methods for anisotropic reflection. This requires storing accumulated power in a directional table for each zone. A proof is given that, subject to certain constraints, only $O(N)$ rays are required for a zonal solution with N zones.

Simulation allows for surfaces which are not zoned to interact with those that are. This is a generalization of the diffuse and specular ray tracing transport work of Malley. This technique can be useful for highly complex or difficult to zone surfaces such as a human face.

The zonal solution methods can be applied to participating media in a fairly natural manner. This zonal method has the benefit of not requiring as much computation time when the medium

is sparse. This also applies to media with anisotropic scattering characteristics, but such a solution requires a large amount of storage.

Wavelength dependent solutions introduce some complications, but can be handled by traditional point sampling techniques. Time dependent solutions are easily handled by image-based solution methods, but are very difficult to apply using zonal methods.

ACKNOWLEDGEMENTS

Thanks to the members of my committee, George Francis, Dennis Gannon, Donald Hearn, and Robert Skeel. To my advisor, William Kubitz, for always being supportive and reading things on even the shortest notice. Thanks to the many people at Illinois for advice and support: Polly Baker, Jenny Barrett, Randall Bramley, Bill Brown, Jean Buckley, Gayanne Carpenter, Herbert Edelsbrunner, Parris Egbert, Brian Foote, George Frank, David Griffith, John Halton, Teresa Hughey, Ray Idaszak, Atul Jain, Ralph Johnson, Henry Neeman, Greg Rogers, Dave Semeraro, Kelvin Sung, Allan Tuchman, Peter Williams. Thanks to the many people in the graphics community who gave me information on their work, and feedback on my own: John Airey, Jim Arvo, Mark Lee, Tom Malley, Claude Puech, Holly Rushmeier, David Salzman, Stephen Spencer, Sam Uselton, and John Wallace.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

This document is about creating realistic looking pictures with computer graphics programs. What is meant by a realistic image? Hall and Greenberg[44] write:

> *Our goal in realistic image synthesis is to generate an image that evokes from the visual system a response indistinguishable from that evoked by the actual environment.*

This is a lofty goal. Attempting to give the viewer *exactly* the same metal impression from a synthetic image as that from viewing the real scene would probably require connecting into the wiring between the eyes and the brain of the viewer. This is not currently feasible.

For practical reasons, we must aim considerably lower than completely duplicating the perceptual response. The primary limitations are that computer generated images are usually displayed in only two dimensions, with limited visual field resolution and color palette, such as on a CRT or printed paper. These limitations also hamper realist painters and photographers. Large field-of-view displays used for flight simulation, and head-mounted displays decrease the limited visual field problem, but their cost and complexity has precluded common use.

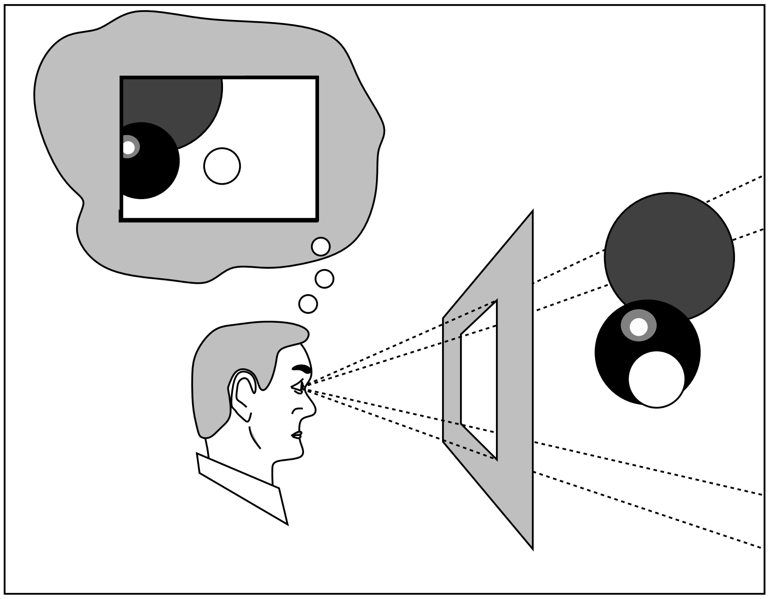

Figure 1.1: A simplified model of mental image processing.

There are two simple ways that these issues are circumvented in computer graphics. The first is shown in Figure 1.1, where a blockade having a rectangular hole is suspended directly in front of one eye of the viewer (the other eye is assumed to be closed). The compromise here is to redefine the objective from duplicating the perceptual response to duplicating the light intensity coming through the rectangular 'window'. The light intensities coming through the hole can be calculated and displayed directly on a CRT. The fundamental problem with this window method is that peripheral vision is ignored; the shift from a continuous full field-of-view 3-D image to a rectangular 2-D image framed by whatever happens to surround a CRT in a terminal room could profoundly change the visual impression of the scene. This simplified viewer model is *not* the same as the full model shown in Figure 1.2. The differences between the simplified and full model cause serious problems in many applications where aesthetic judgements are necessary.

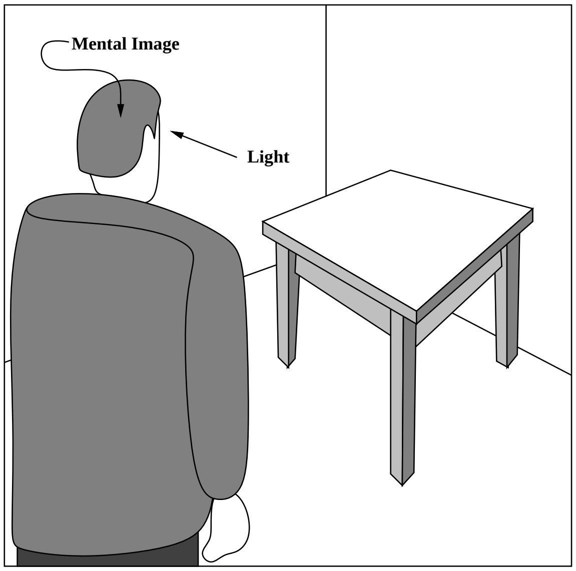

Figure 1.2: A viewer forms an image of a scene.

The other common way to circumvent the rectangular display problem is to eliminate the concept of a viewer by switching to a camera model. If we can calculate the intensities of light that would hit a piece of film in a camera then, in principle, we can determine how a photograph of the scene would appear. Because photographs are usually rectangular, we can display a ‘digitized’ (i.e. discretized) version of the photograph on a CRT. This greatly lessens the peripheral vision issues. They do not go away completely because the surroundings of a CRT will still affect the impressions conveyed by the picture, just as using black or white pages in a photo album can make a difference to the overall perception of the photos in the album. Figure 1.3 shows a pin-hole camera model of image acquisition. The virtual window shown in front of the camera illustrates why it makes little difference whether we are using a window model or a pin-hole camera model. The only major difference seems to be whether our image needs to be spatially inverted before display. Important differences do arise, however, when a finite aperture, which will introduce focusing effects, is used instead of a pin-hole.

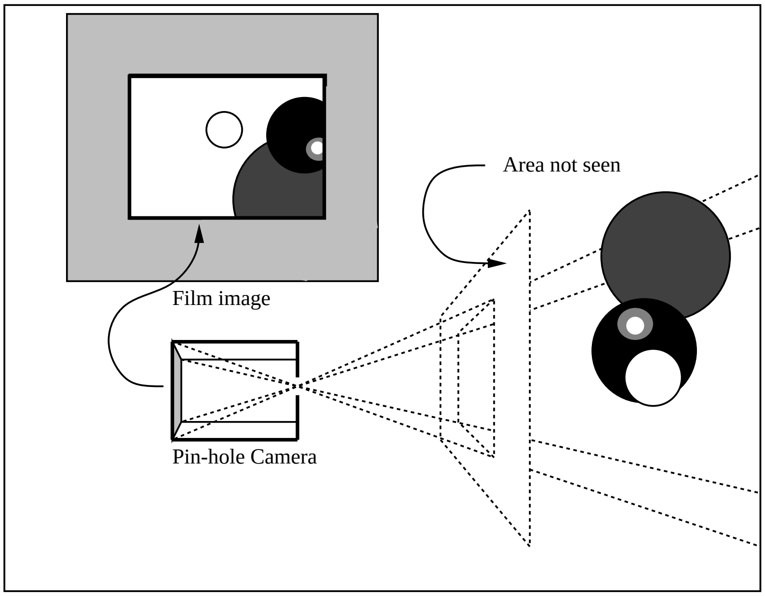

Figure 1.3: A pin-hole camera model of image acquisition.

It could be said that using the camera model is just redefining the problem; the general viewer model is simply too complicated. Thus, instead of creating an approximate impression of looking at a scene, we will create an accurate impression of looking at an approximate photograph of the scene. In spite of this, the camera model will be used for the remainder of this work. This means that the image acquisition problem is reduced to finding what light hits the film for a camera position in a particular scene. The distribution of light hitting the film can be described as the the *image function*, $I(x, y, \lambda)$. The fundamental problem attacked in this document is finding I for all points (x, y) on the film plane, and for all light wavelengths λ in the visible spectrum[1].

It should be emphasized that we have compromised considerably in switching to a camera model from Hall and Greenberg's objective of providing accurate perceptual response. A pho-

[1] If the dye of the film has response curves that include non-visible wavelengths, then we will need to include calculations for those wavelengths. Usually we assume highly idealized models of how the film operates and this issue is ignored.

tograph does generally provide an accurate portrayal of the perceptual impression of a scene, which is the reason photographers are so careful with additional artificial lighting when photographing a scene. Factoring in the characteristics of the human visual system is probably crucial in producing a *realistic* image. Although such perceptual issues are very interesting, and to some extent unexplored in the graphics literature, they are outside the scope of this work. The rest of this document deals primarily with methods for numerically estimating the image function $I(x, y, \lambda)$ for a particular scene and camera position.

This work is a fairly holistic treatment of the image acquisition problem. It begins with fundamentals of optics and surface geometry which are well known in many fields, but are not as widely understood by Computer Graphics practitioners[2]. The machinery of light transport theory is then developed in the language of Illumination Engineering. These fundamentals are then used to justify, and in some cases modify, commonly used local illumination models. Image based methods, which approach image acquisition starting from the eye/film-plane, are then described. Particular attention is paid to image based methods that calculate global illumination effects. Some new ideas on sampling issues in image based methods are also described. Zonal methods, which calculate lighting effects in the scene in a view-independent manner are then described. The zonal method is also extended to allow for general reflectance types in a spatially and temporally efficient manner. The implementational issues involved in global illumination are then discussed. I have attempted to at least partially describe every stage of generating a computed image, and it is especially hoped that the important details of global illumination calculation are fully treated.

[2] This lack of foundational knowledge has been greatly alleviated since the publication of Roy Hall's book[43]

In Chapter 2, some relevant details of optics are presented, with a particular emphasis on surface appearance. The traditional arguments for assuming only the effects of Geometrical Optics are also given. Special attention is paid to the optical characteristics of pigmented surfaces such as paint.

Chapter 3 formulates the mathematics of light transport between surfaces. This yields the Rendering Equation of light transport. This particular formulation is more precise than previous formulations because care is taken here to insure that transparent surfaces such as glass are accounted for in a natural and unambiguous manner. The image generation problem is phrased in terms of an evluation of the Global Radiance Function, which specifies the directional spectral radiance at every point.

In Chapter 4, traditional computer graphics reflection models are examined using the principles of Chapters 2 and 3. A modification of the traditional model for polished surface reflection is also introduced. A new simplification in the model of diffuse transmission is also discussed.

Chapter 5 surveys *image space* methods of finding $I(x, y, \lambda)$. These methods find a solution by working from the image plane out into the scene. The stochastic sampling strategies of distributed ray tracing and path tracing are discussed in a rigorous manner. A method of reducing the number of shadow rays is presented; the collection of all light sources is viewed as one large light source with disconyinuities, and shadow rays are fired toward this 'one' source. This chapter also includes an analysis of static and adaptive sampling strategies, and introduces a new sampling strategy that has some desirable properties, and outperforms Poisson disk sampling for some images. The analysis introduces to computer graphics the concept of *discrepancy*, a measure from numerical integration theory that is useful in evaluating sampling strategies. Discrepancy is particularly useful because it associates a numerical quality estimate to any particular sampling distribution. Readers unfamiliar with Monte Carlo integration may wish to look at Appendix B before reading Chapter 5.

Chapter 6 discusses zonal (radiosity) methods, and pays particular attention to formulating the progressive refinement methods as a physical simulation, rather than as a linear algebraic solution. This idea is used to develop a new zonal technique for non-diffuse surfaces. This chapter also includes a conjecture that we may be able to improve upon the $O(N^2)$ time complexity of traditional radiosity (N is the number of zones). To support this, a proof is given that, subject to certain constraints, a ray tracing solution to a N zone scene can be produced using only $O(N)$ rays. Assuming optimized ray intersection, this means a zonal solution can be produced in some envoronments in $O(N \log N)$ time.

Chapter 7 discusses zonal methods that account for participating media, such as smoke or fog, and a zonal method for arbitrary scattering properties is proposed. It is shown that the physical simulation strategy can be extended to participating media. This solution method has the advantage of being relatively inexpensive for sparse media. Light wavelength and time dependent problems, and methods of solving these problems are also discussed.

The implementational issues of global illumination simulations are discussed in Chapter 8. This discussion centers upon the C++ implementation used to produce the images in this work. It is shown that the local illumination model can be coded as a black box, and the global illumination code can be written without reference to the particular mix of local illumination models to be used.

Finally, Chapter 9 summarizes the work and lists several open questions.

CHAPTER 2

BASIC OPTICS

Almost every computer graphics program models optics in some way. Most make many simplifying assumptions about the physics of optics. In this chapter, the laws of basic optics are presented in a manner oriented toward graphics applications. In Section 2.1, the basic assumptions of physical and geometrical optics are outlined, and the visual impact of discarding physical optics is discussed. Section 2.2 covers the reflection of light from smooth, pure surfaces, primarily in the context of the Fresnel Equations. Section 2.3 discusses the reflection of light from roughened surfaces such as brushed steel. In Section 2.4 more complicated surfaces, such as paint, are described. Translucent materials, such as paper, are described in Section 2.5. Section 2.6 discusses clouds of small particles such as smoke and steam. Finally, the content of the chapter is summarized in Section 2.7.

Those readers with some knowledge of optics may safely skip this chapter (with the possible exception of Section 2.4).

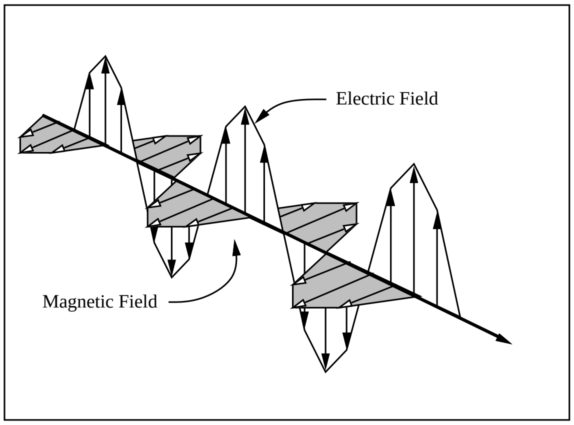

Figure 2.1: Propagation of an electromagnetic wave

2.1 Physical and Geometrical Optics

When trying to understand the behavior of light, physicists usually think of light either as propagating electromagnetic waves or as noninteracting energy carrying rays. The propagating waves have an oscillating electric field and an orthogonal oscillating magnetic field. The wave travels in a direction orthogonal to both fields as is shown in Figure 2.1. The average magnitude of the electric field associated with the wave determines the perceived energy of the wave. In the presence of two waves, the electric field values add (as vectors), so the two waves can interact in complicated ways that affect the perceived energy. If the particle model is used, the interactions are not so complicated because the photons are considered to be independent of each other. That is, the interactions are scalar rather than vector additions. The wave-based model of light behavior is called *physical optics* and the usually simpler ray based model is called *geometrical optics*.

In many situations physical optics effects are not visually important and we can think of the wave as just a ray with an energy (rather than vector field components), so that we can just sum the energy for two interacting rays.

In computer graphics, geometrical optics is almost always used instead of physical optics[1]. This section examines the impact of discarding physical optics in terms of what visible effects will be missing from computer generated pictures. Using geometrical optics allows light behavior to be modeled using many *independent* rays. The primary effects of physical optics that will be missing from pictures stem from interference, diffraction, and polarization.

Interference occurs when overlapping waves have their field values add such that cancellation or reenforcement takes place for a given wavelength. Visible examples of interference include:

- Colored rings in an oil film on a puddle.
- Colored bands on a soap bubble.
- Red and blue regions on a camera lens caused by antireflective coating.
- Bright colors of peacock feathers.
- Bright colors of mother of pearl.

Diffraction occurs when light waves bend traveling around a barrier. Diffraction effects are very important for long wavelength radiation, but are a fairly small effect for visible light. The primary type of visible diffraction is caused by *diffraction gratings*. Diffraction gratings are regular structures with spacings on the order of the wavelength of light. Some animals, such as certain beetles, have diffraction inducing structures that cause very bright colors.

Polarization effects are due to the orientation of the components of the electromagnetic field. For example, the glare of the sun off the pavement has a nonuniform distribution of orientation of field values. Polarized sunglasses preferentially filter these orientations. With the exception of such polarized filters, most examples (not involving glare effects) of visible polarization effects are fairly artificial. However, in some applications, particularly those that involve skylight, polarization can be important to appearance[110].

[1] A notable exception is the work by [75].

material	*refractive index*
Air	1.000293
Water	1.333
Diamond	2.419
Dense flint glass	1.75
Vitreous quartz (glass)	1.45
Salt	1.50

Table 2.1: Indices of refraction for various dielectrics (from Hecht and Zajac[49])

In computer graphics, geometrical optics is usually used because modeling light as waves rather than rays is very difficult, and because of the relative unimportance of the effects of interference, diffraction and polarization. When physical optics effects are modeled they are usually treated as special cases, as will be discussed in Chapter 7.

One physical optics parameter that is important to graphics is the wavelength of light. The wavelength distribution of a set of electromagnetic waves determines its color. This information can be included in geometrical optics by associating a wavelength (or wavelength distribution) with each ray.

2.2 Smooth Surfaces

Smooth surfaces of pure materials (such as smooth glass) reflect light in a well understood way described by the *Fresnel Equations* (Section 2.2.2). These equations predict the portion of light reflected and transmitted at a surface. The equations are usually only written for the interface between two dielectrics[2]. A dielectric is characterized by its index of refraction n. The index of refraction measures how much an electromagnetic wave slows down relative to its speed in vacuum. Typical values for n are listed in Table 2.1.

[2] Dielectrics are essentially non-metals (non-conductors). Semi-conductors are conductors with small extinction coefficient k. Most substances are conductors for at least some wavelengths.

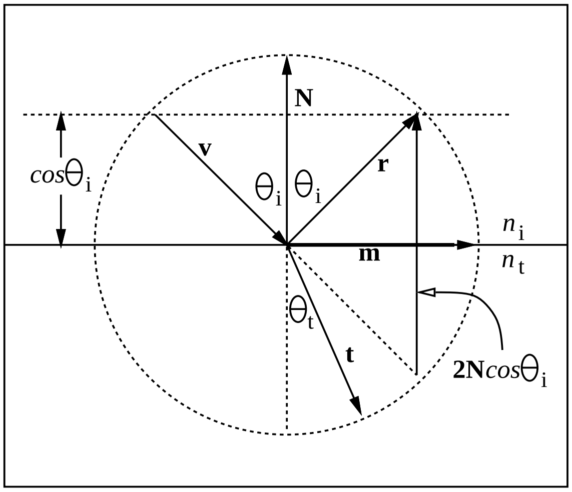

Figure 2.2: Geometry for light hitting a smooth surface.

2.2.1 Snell's Law

When light hits a smooth surface, a portion is reflected, and the rest is transmitted into the object. The direction of the reflected light is given by the Law of Reflection: the angle of the reflection direction **r**, is equal to the angle of incidence θ_i, and will be in the same plane as the incident vector **v** and a surface normal vector **N**. This arrangement is shown in Figure 2.2.

The direction of transmitted light, **t**, is also in the same plane as **v** and **N**, and obeys *Snell's Law*:

$$n_i \sin\theta_i = n_t \sin\theta_t \tag{2.1}$$

When Equation 2.1 cannot be satisfied, all of the light is reflected. Otherwise, a portion will be transmitted. The portions reflected and refracted can be determined using the Fresnel Equations described in the next section.

When implementing graphics simulations, it is convenient to determine **r** and **t** using vector arithmetic. The equation for **r** is fairly simple:

$$\mathbf{r} = \mathbf{v} + 2\mathrm{N}\cos\theta_i = \mathbf{v} - 2\mathrm{N}(\mathbf{v}\cdot\mathrm{N}) \tag{2.2}$$

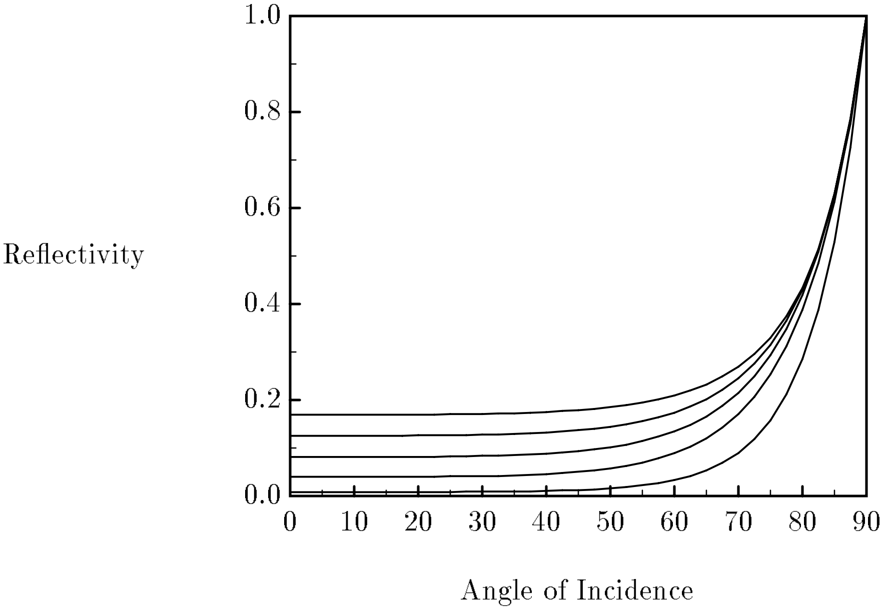

Figure 2.3: Reflectivity of light passing from air to dielectrics of various indices of refraction (1.2, 1.5, 1.8, 2.1, 2.4 bottom to top). Curves were generated directly from Fresnel Equations.

Finding **t** is a bit more tedious. From Figure 2.2, we can see that:

$$\mathbf{t} = -\mathbf{N}\cos\theta_t + \mathbf{m}\sin\theta_t \tag{2.3}$$

This eventually yields[52]:

$$\mathbf{t} = \left[-\frac{n_i}{n_t}(\mathbf{v}\cdot\mathbf{N}) - \sqrt{1-\left(\frac{n_i}{n_t}\right)^2(1-(\mathbf{v}\cdot\mathbf{N})^2)}\right]\mathbf{N} + \frac{n_i}{n_t}\mathbf{v}$$

2.2.2 Fresnel Equations

Conductors, or metals, are described by their refractive index and their extinction coefficient, k. The extinction coefficient of dielectrics is (by definition) zero. Given an interface between a dielectric with refractive index n_i, and a conductor with refractive index n and extinction coefficient k, the reflectivity can be found using the Fresnel Equations[96, 102]:

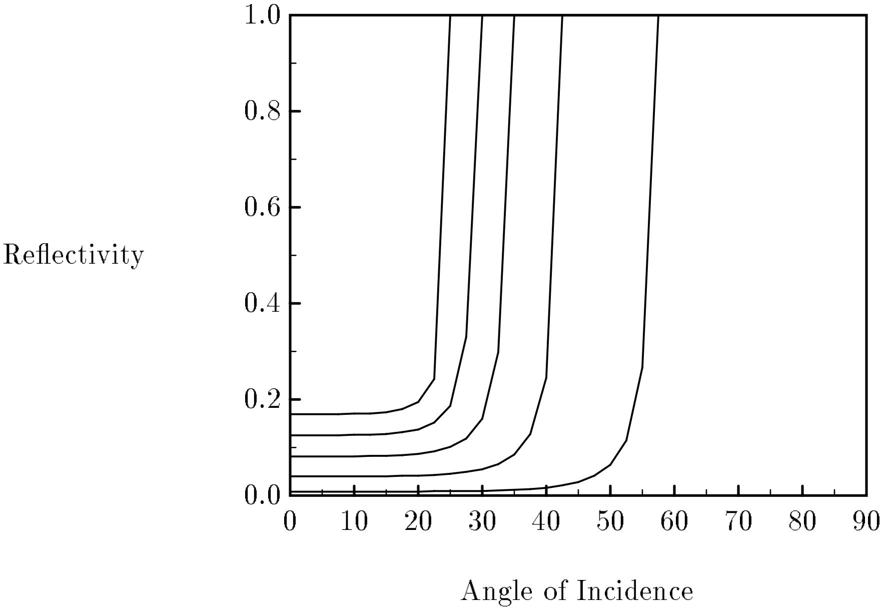

Figure 2.4: Reflectivity of light passing from dielectrics of various indices of refraction (1.2, 1.5, 1.8, 2.1, 2.4 bottom to top) to air. Curves were generated directly from Fresnel Equations.

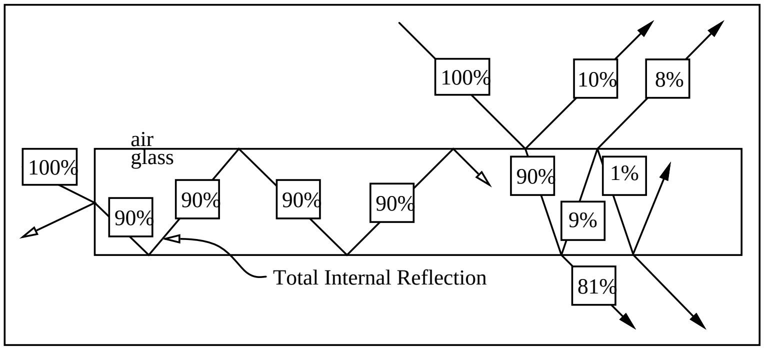

Figure 2.5: Total internal reflection (left) and branching reflection (right).

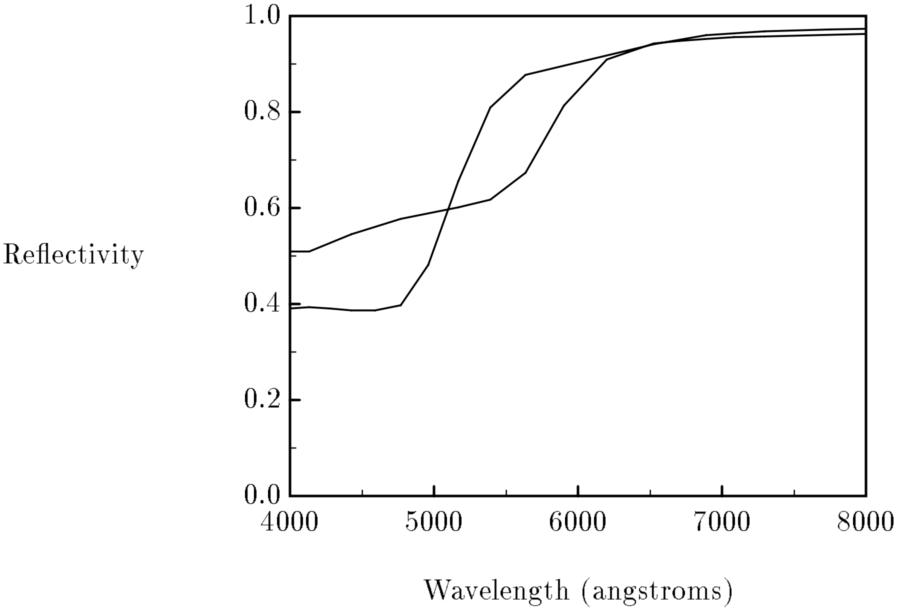

Figure 2.6: Normal incidence reflectivity for various wavelengths for gold (bottom curve at y-axis) and copper. Curves were generated directly from Fresnel Equations using data from Palik [82].

$$R_s = \frac{a^2 + b^2 - 2a\cos\theta + \cos^2\theta}{a^2 + b^2 + 2a\cos\theta + \cos^2\theta} \tag{2.4}$$

$$R_p = R_s \frac{a^2 + b^2 - 2a\sin\theta\tan\theta + \sin^2\theta\tan^2\theta}{a^2 + b^2 + 2a\sin\theta\tan\theta + \sin^2\theta\tan^2\theta} \tag{2.5}$$

Where R_s and R_p are the reflectivities for the two planes of polarization, θ is the angle of incidence, and a and b are given by

$$a^2 = \frac{1}{2n_i^2}\left\{\sqrt{(n^2 - k^2 - n_i^2\sin^2\theta)^2 + 4n^2k^2} + n^2 - k^2 - n_i^2\sin^2\theta\right\} \tag{2.6}$$

$$b^2 = \frac{1}{2n_i^2}\left\{\sqrt{(n^2 - k^2 - n_i^2\sin^2\theta)^2 + 4n^2k^2} - (n^2 - k^2 - n_i^2\sin^2\theta)\right\} \tag{2.7}$$

[82].

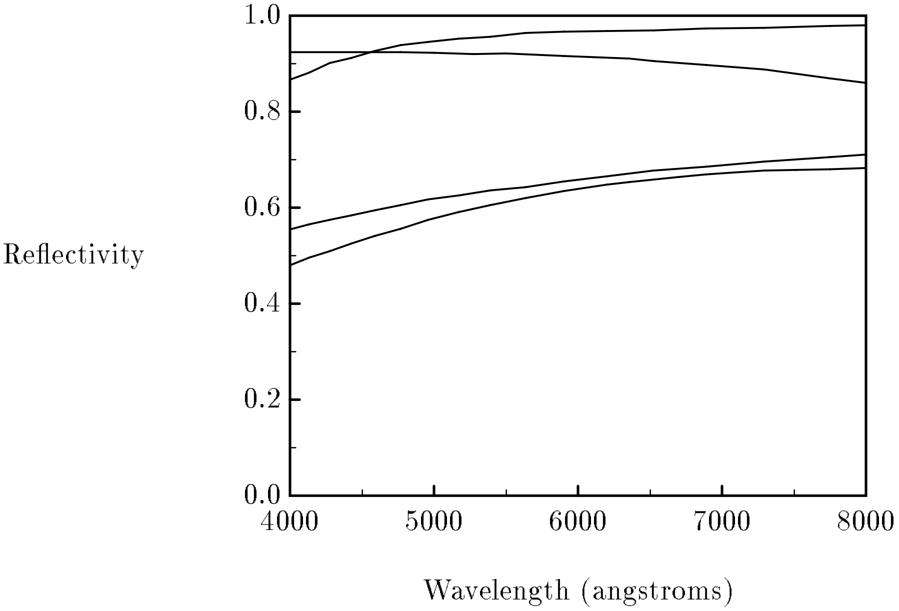

Figure 2.7: Normal incidence reflectivity for various wavelengths for (bottom to top on y-axis) nickel, platinum, silver, and aluminum. Curves were generated directly from Fresnel Equations using data from Palik [82].

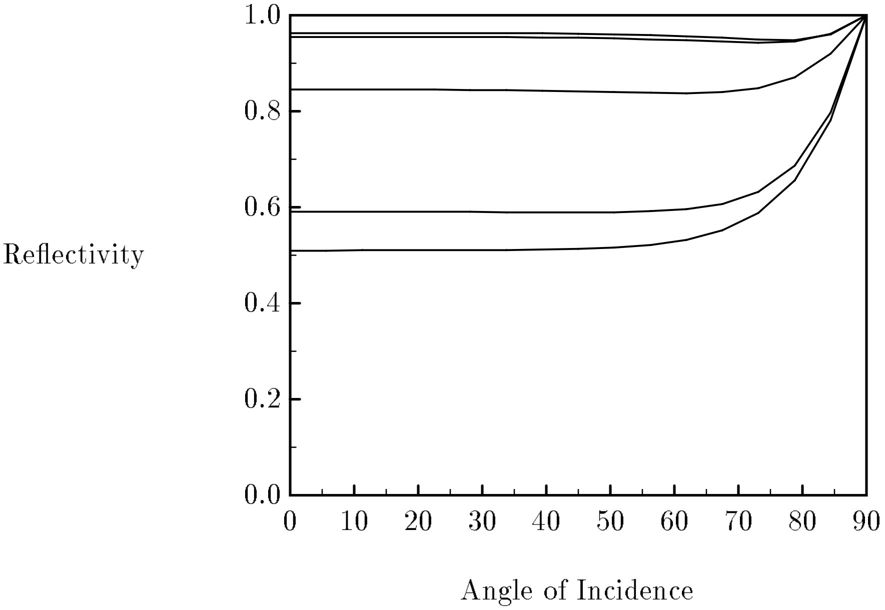

Figure 2.8: Reflectivity of copper for wavelengths 4000, 5000, 6000 7000, and 8000 angstroms (bottom to top). Curves were generated directly from Fresnel Equations using data from Palik [82].

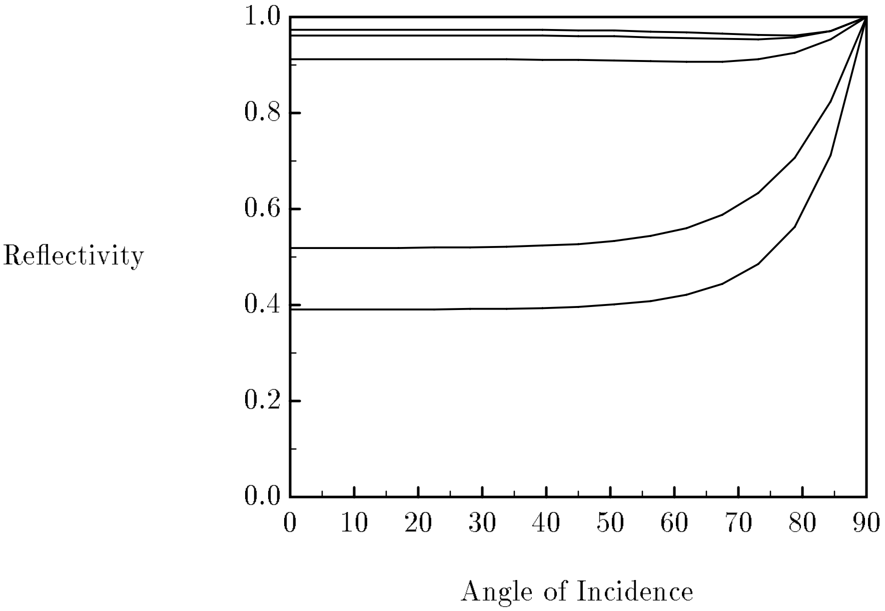

Figure 2.9: Reflectivity of gold for wavelengths 4000, 5000, 6000 7000, and 8000 angstroms (bottom to top). Curves were generated directly from Fresnel Equations using data from Palik [82].

The reflectance for unpolarized light is simply the average of R_s and R_p. The equations for a dielectric-dielectric interface can be found by setting k to zero and using n as the index of refraction for the second dielectric.

Values of n and k can vary with wavelength, so the color properties of materials are treated automatically. Values of n and k for many materials at optical and other wavelengths can be found in Palik [82].

There is an important property of the equations that should be mentioned. First, when θ is 90°, both R_s and R_p go to one for all physical values of the optical constants[3]. This is why we see glare off surfaces when we see them at an acute angle. Another property of the equations is

[3] To see that R_p goes to one, multiply R_p by $\cos^2\theta/cos^2\theta$, or employ l'Hopital's Rule.

that they can be employed without regard to the direction of propagation. This property will become important when ray tracing methods are discussed in Chapter 5.

The reflectivity at an air-dielectric interface is shown in Figures 2.3 and 2.4. Notice in Figure 2.4 that light passing from a dielectric with a higher index of refraction to air will at some angle of incidence reflect completely. This phenomenon is called *total internal reflection*, an example of which is shown on the left of Figure 2.5. Because of total internal reflection, light can travel inside a transparent object for many reflections with almost no attenuation, as happens in fiber optic cables. This property will have an important bearing on the ray tree pruning rules described in Chapter 5.

The refractive index of a dielectric will typically vary somewhat with wavelength, so the reflectivity and angle of transmission can vary for different colors of light. This is why a prism is able to separate a beam of white light into a spectrum. This divergence of transmitted light is called *dispersion*, and will be discussed at greater length in Chapter 7. For metals, both n and k can vary with wavelength. This is why metals can take on 'non-silver' colors. The reflectance curves for two such metals, gold and copper, are shown in Figure 2.6 for normal incidence ($\theta = 0$). The reflectance curves for several 'silver' metals are shown in Figure 2.7. As evident in the figure, these metals are also colored, but not to the extremes of gold or copper. The angular behaviors of the reflectance of gold and copper for several wavelengths are shown in Figures 2.8 and 2.9. As expected, the reflectivity at $\theta = 90°$ goes to one for all wavelengths.

2.2.3 Absorption of Transmitted Light

Once light is transmitted into a medium, it may be absorbed. In a dielectric, this will happen if there are absorptive molecules, such as dye molecules, inside the medium. Dielectrics with an isotropic distribution of absorptive molecules, such as green glass, obey *Beer's Law*:

$$I(\lambda) = I_0(\lambda)e^{-C(\lambda)t}$$

where $I_0(\lambda)$ is the intensity of the light at wavelength λ just after passing into the dielectric, $I(\lambda)$ is the intensity after passing through a distance t of the dielectric, and $C(\lambda)$ is a positive constant that describes the behavior of the dye.

Metals also absorb transmitted light, but they do it without dyes, and do it so quickly that they are opaque for almost any thickness. If the thickness, t, is very small, such as the thin coating of gold film on some sunglasses, then a metal can be partially transparent. The degree of transparency is determined by *Lambert's Law*[4]:

$$I(\lambda) = I_0(\lambda)e^{-4\pi kt/\lambda}$$

where k is the extinction coefficient of the metal at wavelength λ. Lambert's Law can, of course, be thought of as a special case of Beer's Law, where C is based solely on the character of the metal. If there are two adjacent thin films of metal, Lambert's Law can still be used, but a more general form of the Fresnel Equations would be needed to determine boundary reflectivity.

2.3 Rough Surfaces

Most surfaces in real environments are not smooth and of pure composition, so we cannot directly apply the Fresnel Equations to determine a surface's reflective behavior. A great body of both the Heat Transferliterature[106, 102, 80], and the Computer Graphics literature[27, 60, 16, 111] has been devoted to the case when the surface is not smooth, but is made up of a pure material. A close-up view of such a surface is shown in Figure 2.10. Examples of such surfaces

[4] Not to be confused with Lambert's Law of diffuse reflection.

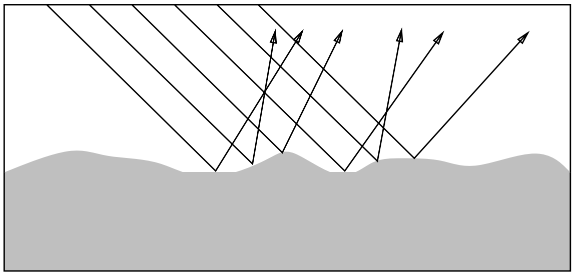

Figure 2.10: Detailed view of a rough surface.

include etched glass and brushed steel. Different ways of treating these surfaces for Computer Graphics purposes are detailed in Chapter 4.

2.4 Composite Surfaces

Most surfaces in nature, and even in man-made environments, are neither smooth nor of pure composition. These *composite* surfaces are made up of at at least two materials in some interesting arrangement. The models of composite surfaces discussed in this section are based largely on the discussion in the books by Rossotti[89], Falk[33], and Williams[122]. More realistic models of composite surfaces allow for varied particle shape and nonuniform particle distribution. Discussion of the subtleties of these general surfaces can be found in Parfitt and Sing[83], and Feller[34].

A classic example of a composite surface is a gloss paint which is made up of some kind of dielectric medium with suspended pigment particles, as shown in Figure 2.11. The pigment particles for a white paint are typically clear dielectrics or 'silver' conductors, with a different index of refraction from the medium. The greater the index of refraction, the greater is the quantity of light scattered at the medium-pigment interface, and the more opaque the paint. Some paints have a medium whose index of refraction changes as it dries, which explains the

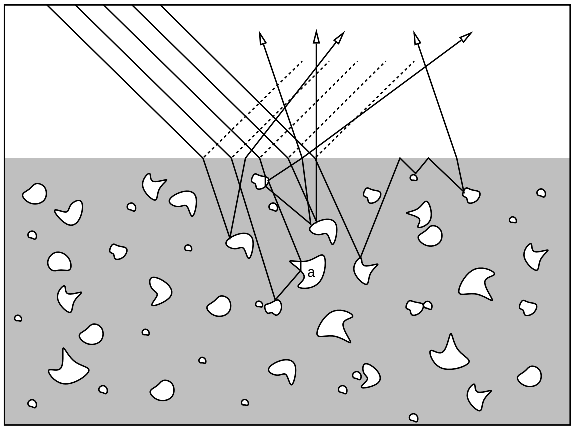

Figure 2.11: Detailed view of a glossy paint.

loss of opacity for some drying paints. In a sense, snow is also a composite surface made up of ice and air. The geometry of the snow flakes on the ground can produce complex reflective behavior[73]. Another good example is an ice cubes with internal air bubbles. If there are many bubbles, the ice becomes opaque; the ice is the medium, and the air bubbles are the pigment particles.

If the pigment particles have a color (caused by dye or positive k), then there will be selective absorption for light traveling through the pigment (such as the light passing through particle 'a' in Figure 2.11), and the paint will be colored. Because the surface of the paint reflects light according to the Fresnel Equations, it will behave as a partial mirror. This mirrorlike behavior accounts for the highlights and reflections seen on gloss painted surfaces.

Many walls are painted with a paint roller or spraygun and have a rough surface. Such surfaces still exhibit mirrorlike behavior, as shown in Figure 2.12, but the reflections may lack coherence, so the highlights will be spread out. Some paints are so rough that they do not look reflective; they are *matte*. Another way paint can look matte is to have the pigment particles pierce the surface of the medium, either by wear or design, as shown in Figure 2.13.

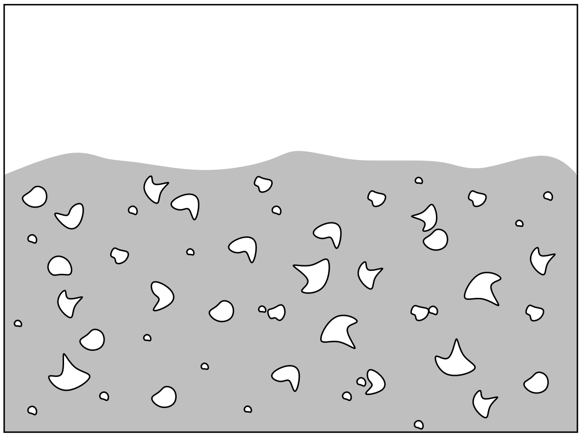

Figure 2.12: Detailed view of a glossy paint with uneven surface.

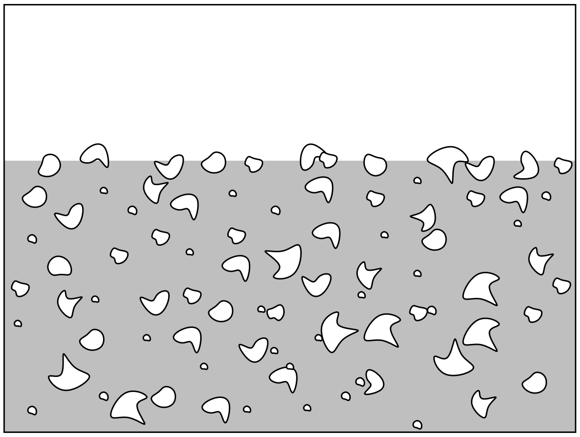

Figure 2.13: Detailed view of a matte paint

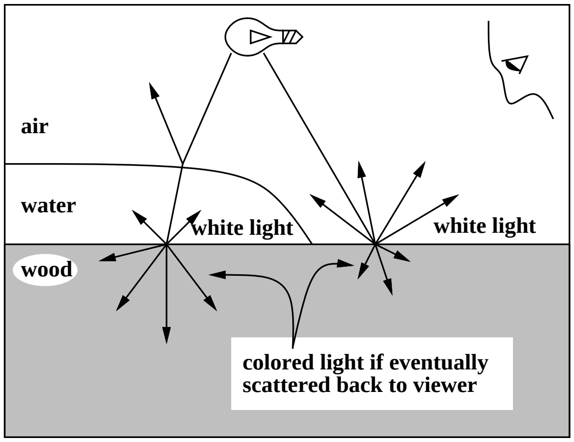

Figure 2.14: Color of wet wood is deeper than color of dry wood

One interesting thing about a polished surface is that its color is usually darker and more saturated than an unpolished surface of the same material. For example, most dark hardwoods get a much deeper color when wet or varnished. This is because the wood particles have an index of refraction that is closer to that of the water than that of air[89]. This means more light will be transmitted from water to the wood than from air to wood, and less light will be reflected at the wood surface. The transmitted light that eventually scatters back out of the wood will have been filtered to be 'wood colored', as opposed to light reflected uniformly across wavelengths at the surface. The light reflected at the water-air interface is reflected specularly (the surface is smooth), so this light is not seen by the viewer (unless the viewer is in the path of reflection, in which case a bright highlight will be seen). This idea is illustrated in Figure 2.14.

2.5 Translucent Surfaces

Some rough surfaces, such as paper, allow some light to pass through. These *translucent* objects are made up of many small particles or fibers that scatter light. They are translucent if their thickness is small enough that a significant amount of light can make it from one side of the

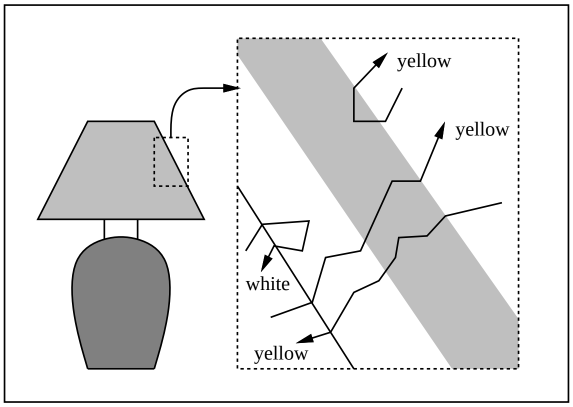

Figure 2.15: Detailed view of a translucent lampshade.

object to the other. As an example, consider a pile of ground glass on a sheet of red paper. Near the center of the pile the glass will appear white because almost all of the light will be backscattered out of the pile before reaching the paper. Near the edges of the pile, where there is less glass to scatter the light, some light will reflect off the red paper, and the red color will show through. This red part of the pile is translucent. The rest of the pile is opaque. Of course, all of the pile will allow *some* portion of incident light to reach the red paper, so opacity is a matter of degree.

A detailed view of a translucent paper lampshade is shown in Figure 2.15. The paper has different reflective properties on the two sides, but should have roughly the same transmittance properties for each side. This is because of the Helmhotz reciprocity rule: The transmittance is a function of the available paths through the shade, and these paths can be traveled in either direction.

2.6 Participating Media

Whenever the environment to be modeled is not associated with a vacuum, the region between surfaces will be occupied by some kind of atmosphere such as air or water. This atmosphere is made up of many molecular particles which can scatter light, and may include larger particles such as dirt, smoke, or steam that also scatter light. A discussion of such effects from a computer graphics perspective, and an excellent treatment of scattering in general, can be found in the paper by Klassen[67]. A more formal treatment can be found in the classic book by van de Hulst[116].

The scattering by large particles can be looked at in terms of geometrical optics. If the particles are small relative to the wavelength of light, then physical optics predicts that light will scatter as if the particle had a larger size. This type of scattering is called *Raleigh Scattering* and is highly biased toward short (blue) wavelengths. Raleigh Scattering is responsible for the blue color of the sky and the color of blue eyes (the eyes have many small particles in suspension). In between classical and Raleigh scattering both physical and geometrical optics are present. This combined type of scattering is called *Mie Scattering*, and the associated theory is complex. All of the scattering can be predicted by associating with each type of particle a scattering cross section and a scattering distribution function.

2.7 Summary

Light can be considered to be rays (geometrical optics), or waves (physical optics). Geometrical optics is usually used in computer graphics because of the independence property of rays. Ignoring physical optics will preclude modeling some observable effects such as interference and diffraction.

Smooth surfaces reflect and transmit light in a well understood way governed by the Fresnel Equations. The equations can be applied when the refractive index and extinction coefficient of a material are known.

Surfaces that are not macroscopically smooth can be better understood by looking at their microscopic structure. Rough surfaces appear matte because of their distributed surface normals. Polished surfaces exhibit mirrorlike behavior at their surface, and materials in or under the polish can scatter the transmitted light in a non-specular way. Thin objects made up of particles or fibers can allow some light to pass through the object, so they exhibit translucent behavior.

Small particles suspended in space can scatter light between surfaces. If the wavelength of light is small relative to the size of the particles, then the scattering can be predicted using geometrical optics. If the wavelength is large relative to particle size, then Raleigh Scattering theory can be used. For intermediate wavelengths, Mie Scattering theory must be used.

CHAPTER 3

GLOBAL ILLUMINATION PROBLEM FORMULATION

One of the consequences of using physical laws for modeling light transport is that we must be at least a little careful about what quantities are used. In Section 3.1, some useful physical quantities are introduced, including *power* and *radiance* which are perhaps the most useful quantities for computer graphics simulations. Section 3.2 considers the radiance at all points and directions in a room to be a function, and frames the computer graphics problem in terms of evaluating this function. Some care is taken to make sure the function is defined at all points including points exactly on surfaces. Some properties of the function are also discussed. Section 3.3 develops the *rendering equation* which describes light interaction at a surface. This development of the rendering equation differs from that of previous authors by including a precise treatment of transparent surfaces[56, 61]. Finally, Section 3.4 summarizes the main points of this chapter.

In the interest of simplicity, this chapter will ignore the possibility of a participating medium such as smoke. This omission will be corrected in Chapter 7.

3.1 Terms and Units

In this section some important terms are defined and described. In particular, the most important physical quantities used in realistic graphics, *power* and *radiance*, are informally introduced. The vocabulary and notation introduced in the section are based closely on the standard terms and notation used in the field of Illumination Engineering. The reason this terminology was chosen over that of the field of Heat Transfer is that the Illumination Engineering community has adopted a standard terminology[57]. This terminology is also commonly used in Physics[31]. This may cause some confusion because much of the Computer Graphics literature uses Heat Transfer notation, notably the groundbreaking papers from Cornell University[40, 23, 24, 56, 93, 117]. Unfortunately, the field of Heat Transfer has no universal standard notation, so it has been avoided in this work. Readers familiar with Heat Transfer notation should note that Heat Transfer's *intensity* is Illumination Engineering's *radiance*.

As discussed in the Chapter 2, the geometrical optics approximation allows us to think of optical transport as noninteracting rays bouncing around a scene that is to be rendered. The fundamental unit that measures the energy of a packet of rays is *radiant energy*, denoted by Q.

The radiant energy is simply a measure of light energy. Since we are interested in the amount of light hitting a surface or film plane during a set time period, *radiant power* (also called *radiant flux*), the radiant energy per unit time, Φ, is often used. Henceforth I will refer to radiant power simply as *power*. Power is often convenient to work with because it allows energy balance constraints to be easily applied[1].

[1] Energy balance constraints are only easy to apply if the solution is steady state or we assume that the speed of light is infinite. Otherwise we have to allow for the lengths of light ray paths. Since the time it takes light to travel across a typical scene (such as a living room) is very small compared to a camera shutter speed or the human temporal visual threshold, assuming infinite light speed is usually appropriate.

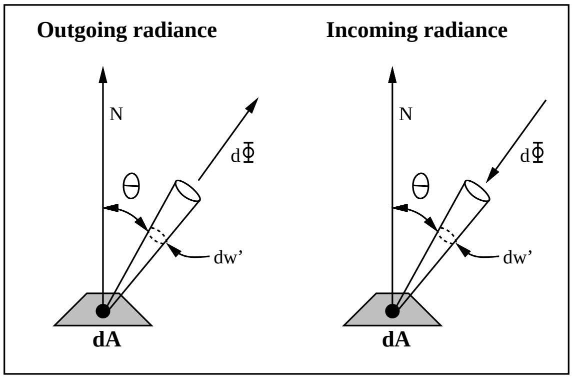

Figure 3.1: Geometry for Definition of Radiance.

Power is sometimes not a natural quantity to use for graphics. For example, we characterize the appearance of an object by its 'color'. Unlike power, this notion of object color can depend on viewing angle, as is true for a mirror. The amount of power traveling from a source in a certain direction can be measured as *radiant intensity*, $I = d\Phi/d\omega$, where ω is the solid angle originating at the source.

One characteristic of radiant intensity is that it depends on the area of the light source. This is not true of object color, which is independent of surface area. The quantity that more closely approximates color is the *radiance*, L (see Figure 3.1):

$$L = \frac{d^2\Phi}{d\omega dAcos\theta}$$

Or in terms of radiant intensity, radiance is:

$$L = \frac{dI}{dAcos\theta}$$

The radiance gives an indication of surface brightness, dependent upon neither the size of the object being viewed, nor the distance to the viewer. It usually suffices to think of radiance as the power passing through a point from a unit solid angle (fraction of the visual field). The radiance at a certain point $\mathbf{x}$ in a direction ψ is denoted $L(\mathbf{x}, \psi)$. A *direction* is just a vector, or can be thought of as a (θ, ϕ) pair in spherical coordinates.

These quantities (energy, power, radiant intensity, and radiance) as described above are taken across many optical wavelengths. Since we eventually want to find visual colors, the wavelength breakdown of these quantities is needed. Instead we can think of them as functions of wavelength by measuring them *per unit wavelength*. These new wavelength dependent quantities are called *spectral energy*, *spectral power*, *spectral radiant intensity*, and *spectral radiance*. Since wavelength dependent information is always needed for graphics, the word *spectral* should be assumed to be implicit for the remainder of the text.

3.1.1 Reflectance Terms and Units

A simple way to describe the reflectance of a surface is by the absolute reflectance $R(\psi_{in}, \lambda)$, the fraction of light at wavelength λ incident from direction ψ_{in} that is not absorbed. The reflectance is often too simple a measurement because the distribution of the reflected light is not described. To overcome this shortcoming, the bidirectional reflectance-distribution function (**BRDF**), ρ, can be used. Written in terms of radiance the expression for the bidirectional reflectance is:

$$\rho(\mathbf{x}, \psi_{in}, \psi_{out}, \lambda) = \frac{dL(\mathbf{x}, \psi_{out}, \lambda)}{L(\mathbf{x}, \psi_{in}, \lambda) d\omega_{in} \cos\theta_{in}} \tag{3.1}$$

Here, $\mathbf{x}$ is the point of reflection, ψ_{in} is the incident direction, ψ_{out} is the direction of reflectance, $d\omega_{in}$ is the differential solid angle the incoming light arrives through, and θ_{in} is the angle between ψ_{in} and the surface normal at $\mathbf{x}$.

The definition of the **BRDF** is not as mysterious as might first appear. The numerator $dL(\mathbf{x}, \psi_{out}, \lambda)$ is the radiance of the surface seen from a point in direction ψ_{out}. The denominator $L(\mathbf{x}, \psi_{in}, \lambda) d\omega_{in} \cos\theta_{in}$ is simply the incident power per unit area. This makes the measurement of the **BRDF** easy in theory. But in practice data for the **BRDF** of real surfaces is hard to come by. Methods of measuring **BRDF** can be rather tedious and expensive, so this lack of data is not surprising[76]. There is some hope that this data shortage is not permanent; an apparatus that will allow cheaper measurement of the **BRDF** of real surfaces has been designed at Lawrence Berkeley Lab, and initial results are promising[91].

An important property of **BRDF**s is called the *Helmhotz Reciprocity Rule*, which states that the **BRDF** is symmetric relative to ψ_{in} and ψ_{out} (REF). Quantitatively, the rule can be stated:

$$\rho(\mathbf{x}, \psi_{in}, \psi_{out}, \lambda) = \rho(\mathbf{x}, \psi_{out}, \psi_{in}, \lambda) \tag{3.2}$$

The reciprocity rule is examined theoretically and verified experimentally in [21].

Sometimes it is more convenient to work with the radiant power Φ than with the radiance L. On these occasions the **BRDF** is cumbersome. It is more natural to view the surface reflection properties in terms of the probability distribution of the reflected light. This can be called the *scattering probability function* (**SPF**), s:

$$s(\mathbf{x}, \psi_{in}, \psi_{out}, \lambda) = \frac{dI(\mathbf{x}, \psi_{out}, \lambda)}{R(\mathbf{x}, \psi_{in}, \lambda) d\Phi(\mathbf{x}, \psi_{in}, \lambda)}$$

The **SPF** directly describes the amount of energy scattered in each direction ψ_{out}. The term $R(\mathbf{x}, \psi_{in}, \lambda)$ appears in the denominator to scale the function to a valid probability density function over the solid angles ω. Thus the probability of an energy packet of wavelength λ incident on point $\mathbf{x}$ from direction ψ_{in} being scattered in direction ψ_{out} is $R(\mathbf{x}, \psi_{in}, \lambda) s(\mathbf{x}, \psi_{in}, \psi_{out}, \lambda) d\omega(\psi_{out})$, and the probability of it being absorbed is $(1 - R(\mathbf{x}, \psi_{in}, \lambda))$. The **BRDF** and the **SPF** have the simple relationship $\rho = Cs\cos(\theta(\psi_{out}))$, where C is the constant that enforces the unit area constraint for a probability density.

3.2 Image and Global Radiance Functions

At a point $\mathbf{x}$ in an environment, such as a room, there is a well defined radiance value for every direction ψ and wavelength λ. This radiance can be denoted $L(\mathbf{x}, \psi, \lambda)$, which for our purposes can be thought of as a function varying over points $\mathbf{x}$ in the scene to be rendered, directions ψ, and visible wavelengths λ. This function is called the *Global Radiance Function*, and it incorporates the movement of all light through an environment. Because light bends at the surfaces of objects (Snell's Law, Section 2.2), L is *not* defined at a point exactly on a surface. This is an easy problem to get around for opaque surfaces, but is more troublesome for transparent surfaces. For this reason, some care will be taken in treating the function on surfaces.

We can look at calculating the image function, I, in terms of evaluating the global radiance function at the viewer position. This is because all of the considerations about the image are present in $L(\mathbf{x}_{eye}, \psi_{view}, \lambda_{vis})$, where $\mathbf{x}_{eye}$ is the eye position, ψ_{view} varies over all directions within the view frustum, and λ_{vis} varies over the visible wavelengths.

The global radiance function has a basic property that is used explicitly or implicitly in most graphics algorithms: L is constant along a line uninterrupted by objects. This property implies that as we change our distance to an object, but not our viewing direction, the radiance we measure does not change. Intuitively, this just means that color does not change with distance, which accords with everyday experience. This property can be formulated as the *Ray Law* (Figure 3.2), which states that L is constant along a ray from $\mathbf{x}$ to the first surface hit by the ray:

$$L(\mathbf{x}, \psi, \lambda) = L(\mathbf{x} - t\psi, \psi, \lambda) \quad \text{if } 0 < t < t_0$$

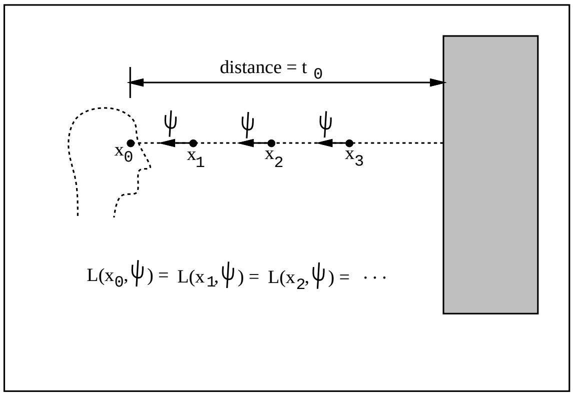

Figure 3.2: The radiance L stays constant along a line of sight.

where t_0 is the distance to the first surface seen from $\mathbf{x}$ in direction ψ (this assumes that ψ is a unit vector). This point is usually determined by sending (tracing or casting) a *ray* from $\mathbf{x}_0$, in direction ψ, until some surface is hit.

Though the idea of the Global Radiance Function sounds elegant, as defined above it is not complete enough to be useful. The Ray Law implies that once we know the radiance at all surfaces, we can 'fill in' the values between surfaces. Unfortunately, the Global Radiance Function is only defined between the surfaces! The most straightforward way to handle this problem is to push the definition of the GRF to surfaces by taking a limit. For opaque surfaces taking this limit is easy because we do not need a radiance to be defined inside the surface. In Figure 3.3 this process is shown, where the radiance at the surface is defined to be the limit as point x is pushed along some path until the distance to the surface is zero. This will insure continuity of *L as measured from above the surface.* This limit is often implicitly assumed when taking about the radiance coming out from a point, or into a point[56, 25].

At a transparent surface, this simple limit idea breaks down. Figure 3.4 illustrates that the limit can be taken from either side of the surface. The limit value will be different on each

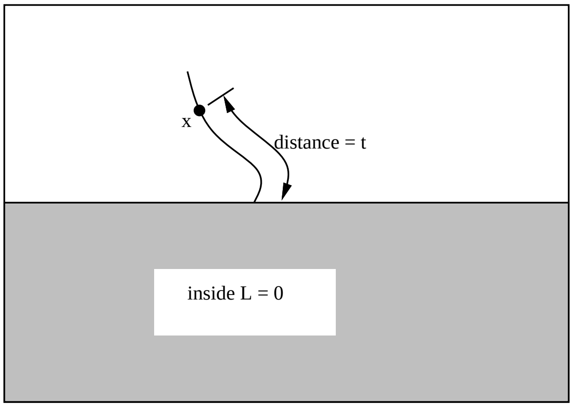

Figure 3.3: The radiance at x can be defined at the surface by taking the limit as t goes to zero.

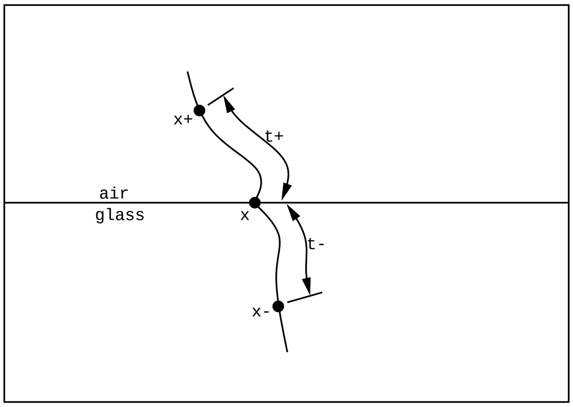

Figure 3.4: The radiance at the surface will be different if the limit is taken from x- instead of x+.

side[2], so L cannot be uniquely defined at the surface without prioritizing 'inside' and 'outside', a concept that does not make sense for the interface between glass and water in a fishtank, for example. One solution would be to always have a point between surfaces when evaluating L at a surface. This point would determine a 'inside' and 'outside' so that the limit could be taken from the side of that point.

[2] The electromagnetic field *is* constant at the surface boundary, which is the basic assumption of the Fresnel Equations. Snell's Law implies that light bends instantaneously at a surface, so its direction of propagation is discontinuous at the surface, though its electromagnetic field maintains continuity.

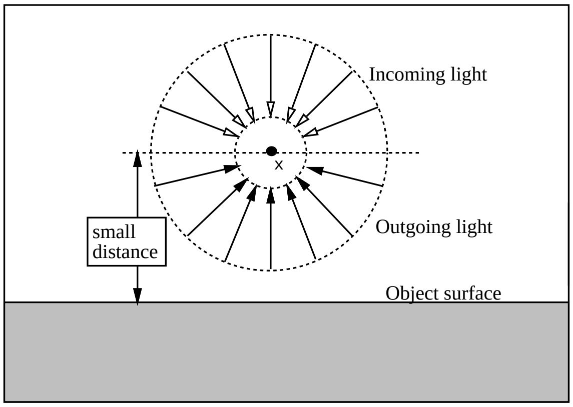

Figure 3.5: The sphere of directions just above a surface can be divided into incoming and outgoing directions.

Another way to take the limit of L at a surface is to divide the set of directions into 'incoming' and 'outgoing' directions as shown in Figure 3.5. In this picture, the black arrows indicate directions that light travels *away* from the surface. The white arrows show incoming directions. The radiance of the surface as measured by a viewer will always be $L(\mathbf{x}, \psi_{out})$, where ψ_{out} is one of the outgoing directions.

On one side of a transparent surface, a viewer will see $L(\mathbf{x}^+, \psi_{out}^+)$ and on the other side the viewer will see $L(\mathbf{x}^-, \psi_{out}^-)$, where ψ_{out}^+ and ψ_{out}^- make up two nonoverlapping hemispheres of directions. We can define the union of these two functions to be the unique limit of L at a surface. This will ignore the radiance values at a surface for *incoming* directions. To describe these values, a separate function L_{in} can be used. L_{in} is defined only at surfaces, and is made up of the incoming directional radiance values not used for L_{out}. This idea is shown visually in Figure 3.6. We could just as easily have used L_{in} for surface values of L, and considered L_{out} as a separate function. L_{out} is a more natural choice because the viewer 'sees' L_{out}, while L_{in} is needed only for reflection and transmission calculations.

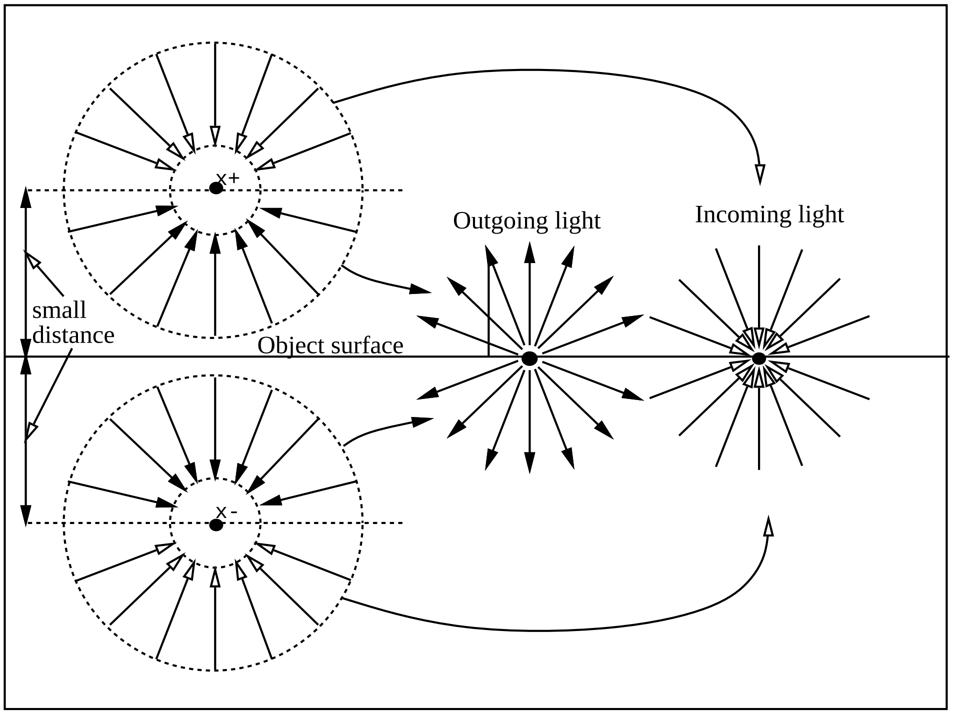

Figure 3.6: Incoming and outgoing radiance distributions at the surface are constructed from piecing together functions on either side of the surface.

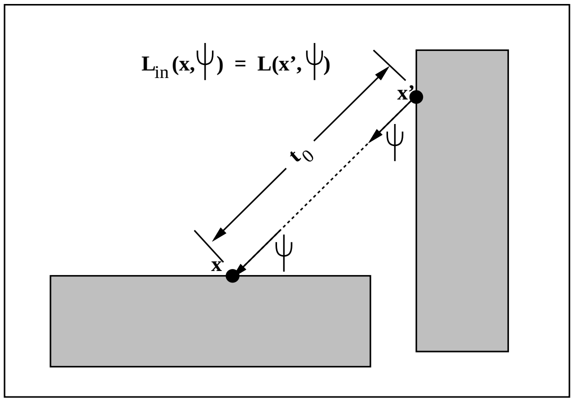

Figure 3.7: The incoming radiance at x is the radiance coming from another point.

Given the concept of L_{out}, we can extend the the Ray Law to include values $(0 < t \leq t_0)$. If $\mathbf{x}$ is not on a surface, then the Ray Law is correct for $(0 \leq t \leq t_0)$. If $\mathbf{x}$ *is* on a surface, then we can write down an expression for L_{in}:

$$L_{in}(\mathbf{x}, \psi, \lambda) = L_{out}(\mathbf{x} - t_0\psi, \psi, \lambda)$$

Here t_0 is again the distance to the first point seen from $\mathbf{x}$ in direction $-\psi$. The geometry of this idea is shown in Figure 3.7. Because L at surfaces is defined to be L_{out}, the equation could be also be written:

$$L_{in}(\mathbf{x}, \psi, \lambda) = L(\mathbf{x} - t_0\psi, \psi, \lambda)$$

This equation is most of the basis for most computer graphics lighting models. In the next section, adding some reflection properties to the surface will produce the rendering equation that forms a model for almost all computer graphics illumination methods.

The continuity of L depends on the continuity of the surfaces and on the continuity of the light emitted from surfaces. Assuming that the light emittance distribution functions and light reflectance distribution functions are piecewise continuous, and that the surfaces themselves are piecewise continuous, then L will also be piecewise continuous. This can be seen by examing the continuity of $L(\mathbf{x}, \psi)$ holding each variable constant in turn. Holding the point,$\mathbf{x}$, constant, L is the radiance seen in all directions from that point. This will be the radiance of surfaces seen from $\mathbf{x}$. Except at surface boundaries, these radiances will be piecewise continuous because the surface (and its reflectance and emittance) is piecewise continuous. Holding the direction, ψ, constant and varying the point, $\mathbf{x}$, also 'pans' across surfaces, so the same reasoning applies. Varying both variables in a continuous manner also 'pans', so the Global Radiance Function L is piecewise continuous. This continuity condition will not hold for fractal surfaces, but will apply to discretizations of such surfaces.

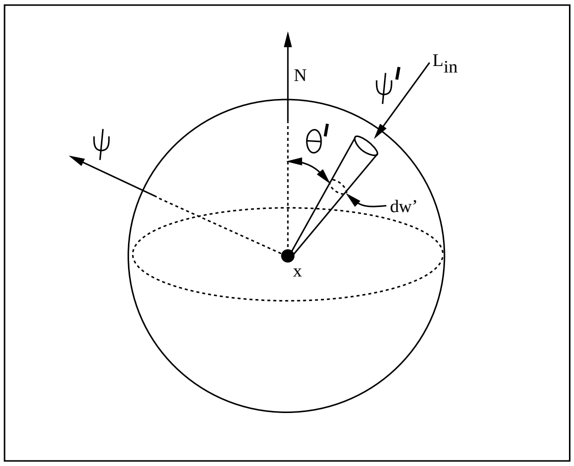

Figure 3.8: Geometry for Directional Rendering Equation.

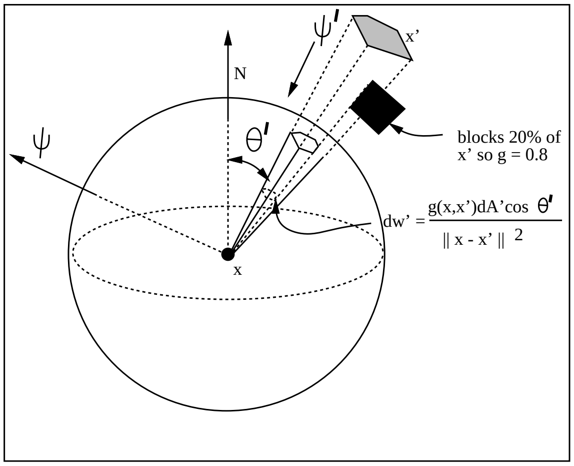
ψ'
x'
N
blocks 20% of x' so g = 0.8
ψ
θ'
dw' = g(x,x')dA'cos θ' / || x - x' || 2
x

Figure 3.9: Geometry for Pointwise Rendering Equation. The black patch occupies 20% of the solid angle subtended by x'.

3.3 Rendering Equation

In the last section we saw that finding the value of $L(\mathbf{x}, \psi, \lambda)$ at a point $\mathbf{x}$ *not* on a surface is accomplished by finding the identically valued $L_{out}(\mathbf{x}_0, \psi, \lambda)$, where $\mathbf{x}_0$ is the point on the surface seen from $\mathbf{x}$ in direction $-\psi$. This can be further specified by dividing the surface radiance into reflected and emitted components:

$$L_{out}(\mathbf{x}_0, \psi, \lambda) = L_e(\mathbf{x}_0, \psi, \lambda) + L_r(\mathbf{x}_0, \psi, \lambda)$$

Here L_e is the emitted component of spectral radiance, and L_r is the reflected component. L_e is presumably known from input data, so the main computational problem is finding L_r. Since all the reflected light must come from the set of all incoming directions $\psi \in \bigcirc$, the expression for L_r comes straight from the definition of ρ (Equation 3.1). If direction ψ' is the only source of non-zero radiance, then,

$$L_r(\mathbf{x}, \psi, \lambda) = \rho(\mathbf{x}, \psi, \psi', \lambda) L_{in}(\mathbf{x}, \psi', \lambda) \cos\theta' d\omega'$$

If there are many incoming directions with non-zero radiance, then we must integrate over all incoming directions[3] that influence L_r. This yields:

$$L_{out}(\mathbf{x}, \psi, \lambda) = L_e(\mathbf{x}, \psi, \lambda) + \int_{\psi' \in \bigcirc} \rho(\mathbf{x}, \psi, \psi', \lambda) L_{in}(\mathbf{x}, \psi', \lambda) \cos\theta' d\omega' \tag{3.3}$$

The geometry for Equation 3.3 is shown in Figure 3.8.

The rendering equation is often called the *transport equation* in the heat transfer literature[54]. The recursive nature of this equation makes it non-trivial to solve. The form of Equation 3.3 is most similar to Immel et al.'s formulation of the rendering equation[56]. Kajiya's form differs in that it integrates over all surfaces rather than angles, and it is not written in terms of radiance[61]. Modifying Kajiya's representation to use radiance yields:

$$L_{out}(\mathbf{x}, \psi, \lambda) = L_e(\mathbf{x}, \psi, \lambda) + \int_{\text{all } \mathbf{x}'} g(\mathbf{x}, \mathbf{x}') \rho(\mathbf{x}, \psi, \psi', \lambda) L_{in}(\mathbf{x}, \psi', \lambda) \cos\theta' \frac{dA' cos\theta''}{\|x' - x\|^2} \tag{3.4}$$

Here x', with differential area dA', varies over all points of all surfaces, $g(\mathbf{x}, \mathbf{x}')$ is a geometry term that is one if x' can 'see' x and zero otherwise, and ψ' is the direction from x' to x. The geometry for this equation is shown in Figure 3.9.

[3] This means the definition of ρ must be extended to include all directions, including those below the plane of scattering.

Because the radiance arriving at $\mathbf{x}$ from $\mathbf{x}'$ is *outgoing* from $\mathbf{x}'$, Equation 3.4 can also be written:

$$L_{out}(\mathbf{x}, \psi, \lambda) = L_e(\mathbf{x}, \psi, \lambda) + \int_{\text{all } \mathbf{x}'} g(\mathbf{x}, \mathbf{x}')\rho(\mathbf{x}, \psi, \psi', \lambda) L_{out}(\mathbf{x}', \psi', \lambda) \cos\theta' \frac{dA' cos\theta''}{\|x' - x\|^2} \qquad (3.5)$$

This form of the equation is convenient because it is expressed only in terms of the L_{out} of the surfaces.

Both of the equations above are of the form:

$$a(x) = b(x) + \int_{x' \in \Omega} k(x, x') a(x') d\mu(x')$$

Equations of this form are *Fredholm Equations of the Second Kind.*

3.4 Summary

The physical quantities power and radiance are useful for graphics. Radiance is used when the lighting at a point is considered because it is an intensive quantity. Power is used when energy transport or energy balance is of concern.

The global illumination problem can be thought of as a function evaluation problem, where the function is the Global Radiance Function, L, which is defined at all points and in all directions. The value of L at surfaces can be defined to be the outgoing radiance, L_{out}. The incoming radiance at a surface can be defined as a separate function L_{in}. This distinction is needed only because of transparent surfaces.

The Global Radiance Function obeys the Ray Law; the radiance is constant along a line of sight between objects. The Global Radiance Function is piecewise continuous if the surfaces in scene are themselves piecewise continuous.

On the basis of the Ray Law and the definition of **BRDF** , we can write an expression for the radiance of a point in terms of the radiances of other points. This *rendering equation* is a Fredholm Equation of the Second Kind, and can be written down as an integral over all directions seen from a point, or as an integral over all points that lie on surfaces.

CHAPTER 4

SURFACE REFLECTION MODELS

Although all surfaces obey Snell's Law at a *microscopic* level, complex small-scale structure can give rise to *macroscopic* reflection properties that are quite complex, as illustrated in Chapter 2. This macroscopic behavior can be described by the **BRDF** for that surface.

The two crucial parameters in the rendering equation derived in the last chapter are the distribution of surfaces, and the characteristics of the **BRDF** for the surfaces. In this chapter, several types of idealized **BRDF**s are discussed in the context of traditional computer graphics reflection models. The relationship of these models to the surface geometry classes presented in Chapter 2 is also discussed.

Section 4.1 describes the behavior of perfectly diffuse *Lambertian* surfaces, which are an idealized form of matte surfaces. That section also discusses diffuse transmitters, an idealized type of translucent surface. A new modification to the diffuse transmission model is also made in that section. In Section 4.2, perfectly smooth reflecting surfaces are examined, based on the discussion of Section 2.2. Surfaces that are part diffuse and part mirror are discussed in Section 4.3. This section presents an improved model of how such surfaces reflect light by including Fresnel Equations effects for the specular *and* diffuse terms. Section 4.4 describes reflection behavior that is more general than diffuse and specular behavior. In Section 4.5, the

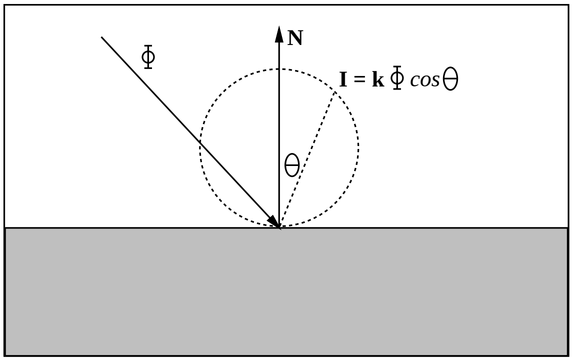

Figure 4.1: Ideal diffuse reflection.

principles described in the earlier parts of this Chapter are used to form guidelines for good parameter selection in traditional lighting models. Finally, Section 4.6 summarizes the content of this chapter.

4.1 Diffuse Reflection and Transmission

Although the reflectance characteristics of real surfaces are quite varied and complex, there are a few simple surface types that usually are used to approximate real reflection distributions. The most common surface type used in graphics is an idealized matte surface, the diffuse reflector. The diffuse reflector is sometimes called a *Lambertian* surface because it obeys Lambert's Law, which states that the **BRDF** is constant. The **BRDF** can only be constant if the numerator in Equation 3.1 is constant. This implies that a diffuse reflector has a constant spectral radiance at all viewing angles under steady lighting conditions. We can see that this is a decent approximation for many materials such as matte paint, which do not noticeably change *color* (the perceptual approximation to spectral radiance) as we change our viewpoint. This means the diffuse **BRDF** , ρ_d is given by the constant:

$$\rho_d(\psi_{out}, \psi_{in}, \lambda) = \frac{R(\lambda)}{\pi}$$

And the corresponding scattering distribution function, as shown visually in Figure 4.1, is:

$$s_d(\psi_{out}, \psi_{in}, \lambda) = \frac{\cos\theta}{\pi}$$

Of course, these equations for ρ_d and s_d are valid only for light incident and reflecting *above* the plane. Otherwise both ρ_d and s_d are zero.

The radiance for a diffuse reflector can be related to the power hitting it at point $\mathbf{x}$:

$$L(\mathbf{x}, \lambda) = \frac{R(\mathbf{x}, \lambda)\Phi(\mathbf{x}, \lambda)}{\pi A} \tag{4.1}$$

Here $\Phi(\mathbf{x}, \lambda)$ is the power hitting the surface and A is the area of the surface. This is a useful property because it is sometimes convenient to work in units of power, and to switch later to radiance.

4.1.1 Diffuse Transmission

Some materials, such as paper, exhibit approximately diffuse reflection[1], but also allow some transmission of light. This transmission does not allow the light to pass unscattered, so detail is lost coming through the material. This type of transmission is often called *translucence*. Translucent materials can be modeled as surfaces which diffusely reflect a portion of incoming light, absorb another portion, and diffusely transmit the rest. For light coming from 'above' such a surface, the **BRDF** can be expressed:

$$\rho_t(\psi_{out}, \psi_{in}, \lambda) = \begin{cases} \frac{R(\lambda)}{\pi} & \text{if } 0° < \theta < 90° \\ \frac{T(\lambda)}{\pi} & \text{if } 90° < \theta < 180° \end{cases}$$

The corresponding **SPF** , as shown in Figure 4.2, is:

$$s_t(\psi_{out}, \psi_{in}, \lambda) = \begin{cases} \frac{R(\lambda) cos\theta}{(R(\lambda)+T(\lambda))\pi} & \text{if } 0° < \theta < 90° \\ \frac{-T(\lambda) cos\theta}{(R(\lambda)+T(\lambda))\pi} & \text{if } 90° < \theta < 180° \end{cases}$$

[1] Paper, and many other materials, have strong directionally dependent reflection at extreme angles.

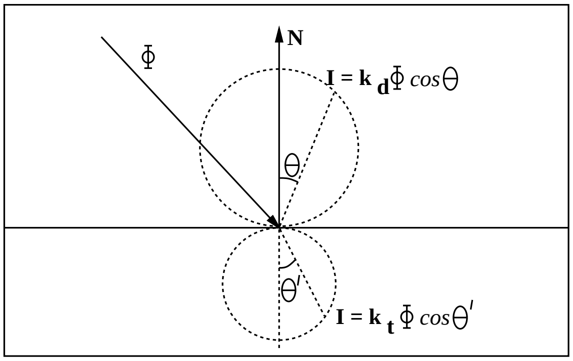

Figure 4.2: Ideal diffuse transmission.

If the incident light comes from 'below' the surface, then we will have the same situation, but in general we might have R' and T' for these angles that are different from R and T. Because T and T' are determined by the 'available paths' that light can take through the material, the reciprocity principle implies that $T = T'$. This observation reduces the two directional transmission coefficients to one common coefficient. If the material is homogeneous in structure (uniform distribution of fibers or grains), then we'd also expect $R = R'$. This will not be the case if one side of the material is somehow different than the other, as is the case for paper that has printing on it.

4.2 Specular Reflection and Refraction

A very commonly used reflective type is the specular surface. Specular surfaces include polished metals, glasses, and any smooth reflective surface. The specular surface is most easily described by its **SPF** which is a delta function times an attenuation term. The attenuation will typically vary with incident angle as discussed in Section 2.2. This is evident in the extreme reflectivity of glass viewed at an acute angle, and its transparency when viewed straight on. As implied by

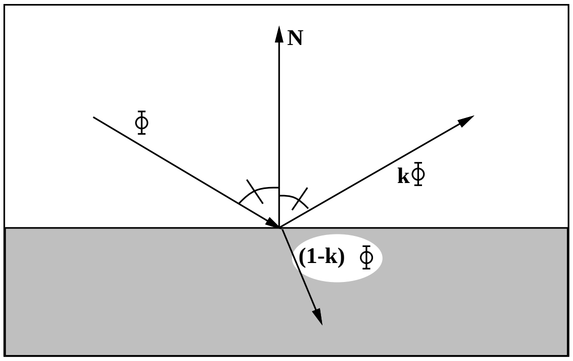

Figure 4.3: Reflection from an ideal specular surface.

Figure 4.3, the specular **SPF** is:

$$s_s(\psi_{out}, \psi_{in}, \lambda) = k_s \delta(\psi_{\mathbf{s}} - \psi_{\mathbf{out}}) + k_t \delta(\psi_{\mathbf{t}} - \psi_{\mathbf{out}})$$

Here k_s and k_t are determined by the Fresnel Equations, δ is the delta function (see Glossary), $\psi_{\mathbf{s}}$ is the direction in which light incident from ψ_{in} reflects in, and $\psi_{\mathbf{t}}$ is the direction light incident from ψ_{in} refracts. If the surface is a metal, the transmitted light will be quickly absorbed.

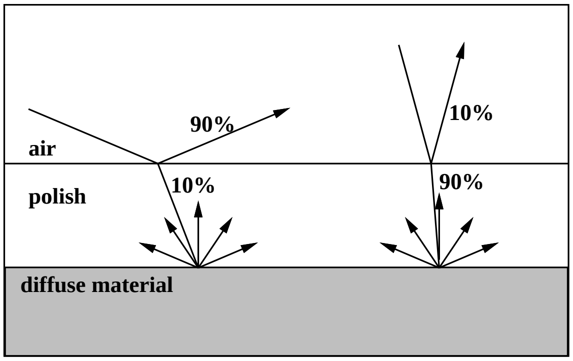

Figure 4.4: Simplified model of polished reflection.

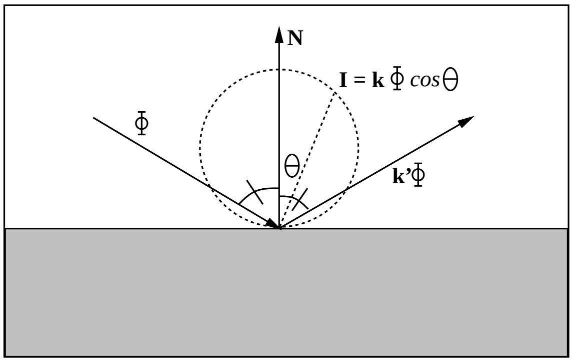

Figure 4.5: Diagram of energy reflection from combined surface.

4.3 Combined Diffuse and Specular Reflection

Many graphics algorithms use a **BRDF** model that is a linear combination of diffuse and specular[121, 44]. If we call the **BRDF** for diffuse ρ_d and the specular **BRDF** ρ_s, the combined **BRDF** ρ_c is:

$$\rho_c = k_d \rho_d + k_s \rho_s$$

This reflection model is usually used to approximate 'polished' surfaces, such as varnished wood. This type of reflector can be thought of as a diffuse reflector covered with a thin dielectric coating. This coating will obey the Fresnel Equations, so it will have variable reflectivity at grazing angles, as shown in Figure 4.4 and Figure 4.5. To account for this variable specularity, k_s can first be calculated using the Fresnel Equations and the refractive index of the polish, and the **BRDF** will be:

$$\rho_c = [1 - k_s(\theta)]\, k_d \rho_d + k_s(\theta) \rho_s$$

This expression allows k_d and k_s to vary with angle in a natural manner, and allows glare effects found in real scenes. This improvement means that k_s will not have to be 'tuned' if the user wants to model the extreme reflection of polished surfaces when θ is near 90°.

4.4 Glossy Reflection

There are some surfaces that are *not* adequately approximated by diffuse, specular, or combined models. The primary example is brushed metal, which shows some 'fuzzy' reflection. The **BRDF** of these so called *glossy*[26] surfaces can be modeled in several ways.

If the microscopic structure of the surface is known, an analytical model can be used to find the **BRDF** , as was done to derive the Torrance-Sparrow model of reflection[27, 111]. The **BRDF** can also be found by simulated experiment: a model of the surface can be constructed, and the **BRDF** can be approximated by sending many rays from various directions and observing reflections[16].

If the microscopic structure of the surface is not known, then observed values can be used, if available. Otherwise, an empirical model, such as the 'Phong **BRDF** ' used by Immel et al. could be used[56]. It has never been demonstrated that such empirical functions are not adequate for computer graphics applications. In fact, it *has* been demonstrated that empirical functions are sufficient in some contexts. Amanitides' very simplified model of reflection produced excellent pictures of rough metal[3].

It may be much more important to capture surface directional grain than to model a 'accurate' rough reflection function. These *anisotropic* reflection models have been explored by Kajiya[60] and Poulin and Fournier[87].

4.5 Standard Computer Graphics Reflection Models

In many computer graphics programs (such as the Hall model[44]), color is calculated by an equation similar to:

$$C = k_d C_l(\mathbf{N} \cdot \mathbf{L}) + k_s C(\psi_s) + k_t C(\psi_t) + k_h C_l(\psi_s \cdot \mathbf{L})^n$$

Where:

C	Color of surface.
C_l	Color of light source.
$\mathbf{N}$	Unit surface normal vector.
$\mathbf{L}$	Unit vector toward light source.
ψ_s	Unit vector in reflection direction.
ψ_t	Unit vector in transmission direction.
k_d	Diffuse reflectance.
k_s	Specular reflectance.
k_t	Specular transmittance.
k_h	Phong reflectance.
n	Phong exponent.

Here it is assumed that the light source is a 'point light source infinitely far away'. This allows us to use a constant C_l at all points (L_{in} is a constant). The terms each account for a different effect:

$k_d C_l(\mathbf{N} \cdot \mathbf{L})$	Diffusely reflected light.
$k_s C(\psi_{\mathbf{s}})$	Specularly reflected light. $C(\psi_s)$ is the color seen in direction ψ_s, which is attenuated by the specular reflectance k_s.
$k_t C(\psi_{\mathbf{t}})$	Specularly transmitted light. $C(\psi_{\mathbf{t}})$ is the color seen in direction $\psi_{\mathbf{t}}$,

which is attenuated by the specular transmittance k_t.

$k_h C_l(\psi_{\mathbf{s}} \cdot \mathbf{L})^n$ Phong highlight term.
Gives fuzzy reflection of point light. The Specular term would take care of this for non-point light sources.

For smooth metals, k_s should be the only non-zero constant, and should be set to the normal reflectance of the metal. If ray tracing is not available (so k_s and k_t cannot be used), then k_d and k_h can be set to the normal reflectance of the metal. For clear dielectrics, k_s and k_t should be non-zero constants. If the object is to be viewed from a grazing angle, k_s should be made larger if Fresnel Equations cannot be used. The sum of k_s and k_t should be one. The k_h term can be used if only point light sources are available. For matte surfaces, k_d should be non-zero. For polished surfaces, k_s should also be non-zero.

The chief problems with this model are that there is no global illumination and that k_s does not obey the Fresnel Equations. This means indirect lighting will not be included, and glare will look wrong. Some codes do use the Fresnel Equations[43], but only for pure dielectrics; polished surfaces are restricted to constant k_s.

4.6 Summary

The macroscopic reflectance properties of surfaces can be described by their **BRDF** . The diffuse surface is an approximation to a matte material. The diffuse transmitter is an approximation to the translucent surface, and it was argued that transmission characteristics should be the same in both directions. That observation simplifies the use of the diffuse transmitter model by eliminating one parameter. The specular surface is a smooth dielectric or metal. The combined surface is an approximation to polished materials, and thus the specular term should follow the Fresnel Equations, and the diffuse term should only reflect light not reflected by the specular term. This angle based interdependence between the diffuse and specular term is not used in previous reflection models, but is vital for correct glare effects. Any reflection behavior not covered by these approximations is called glossy, and such surfaces can have arbitrarily complex **BRDF**s . Standard parametrically driven Computer Graphics reflection models are not sufficient for realistic behavior, but guidelines were given that at least approximate desired behavior.

CHAPTER 5

IMAGE SPACE SOLUTION METHODS

Traditionally, most graphics programs produce pictures by determining a color value for each pixel of a raster screen. This is done in two steps: finding out which surface is visible through that pixel, and finding the radiance coming from the surface toward the pixel. This class of methods operates by finding only those radiances that contribute to the image. In some sense this is really solving for I by using lazy evaluation of the Global Radiance Function; we find radiances at only those locations that are visible. In this chapter these *image based* methods will be described, and some extensions of previous techniques will be shown.

In Section 5.1, the concept of the image function, I, will be expanded, and weighted area averaging techniques of converting I to a discrete (raster) image will be discussed. That discussion also includes some guidelines for selecting filters (weighting functions) that disallow many standard filters such as the cone, pyramid, sinc, and nice. A new filter that does satisfy the guidelines, while maintaining some of the good characteristics of the standard filters, is also presented.

Section 5.2 outlines the use of direct lighting and the ambient term in traditional computer graphics. Whitted-style ray tracing is presented in Section 5.3. More modern stochastic ray tracing methods, including a careful review of the basic mathematics behind them, are presented in Section 5.4. A new method of shadow ray optimization, where only one shadow ray is sent for each viewing ray, is also described.

In Section 5.5, methods of static sampling (where the number of samples in a pixel is predetermined) are reviewed and compared. This discussion differs from that of previous authors in Computer Graphics because the machinery of Integration Theory, rather than Signal Processing, is used to predict sampling performance. In addition, a new static sampling method that has several advantages over even Poisson Disk sampling is presented. Adaptive sampling methods are reviewed in Section 5.6, and a new adaptive strategy is presented. That section also argues, against prevailing wisdom, that hierarchical sampling *cannot* be straightforwardly applied to Distributed Ray Tracing because of the peculiarities of the sampling space used when performing Uncorrelated Jittering.

In this and the next two chapters, issues of color will be ignored. The implications of adding wavelength dependencies will be addressed in Chapter 7.

5.1 The Image Function

As outlined in Chapter 1, we can create an image using a viewer model (Figure 1.1) or camera model (Figure 1.3). Ultimately, we will display the image on a device, or generate hardcopy using a film recorder or color printer. Almost all display devices we might use are digital and represent pictures with a rectangular lattice of *pixels* (short for picture elements). To set values for these pixels, most devices use one number (three or four values for color systems).

Assuming we want to produce a greyscale image for a digital display device, we have to create a digital image, specified by a number (usually one byte long) for each pixel. In other words, we need to specify all values $P(x_i, y_j)$, where i and j are row and column numbers on the device. Currently, a high-end RGB monitor will be 2048 by 1536 or 1280 by 1024. These are some of the few numbers *not* going up explosively in the computer industry. A color film recorder or color printer will often have greater resolution, with up to 200 points per inch for a thermal color printer, and up to 800 points per inch for a film recorder[36].

Assuming we have a pin-hole camera model specified by a pin-hole location and a film-plane, then we first wish to find the radiance at the pin-hole seen from each spot on the film. This radiance determines the film response at each point, and can be described by the image function $I(x, y)$, where x and y are coordinates on the plane. Usually the film is assumed to be perfectly linear in response, with infinite resolution. Assuming I is known, then we can set pixel values:

$$P(x_i, y_j) = f_{ij}(I(x, y))$$

Where f_{ij} is a function that operates on I. Usually one function f is used for all f_{ij}, and f typically is the integral of a weighting function w (centered at (x_i, y_i)) multiplied by I:

$$P(x_i, y_j) = \int w(x - x_i, y - y_i) I(x, y) dA$$

Here the of area integration is wherever the weighting function is nonzero. This region is called the *support* of w. Because we do not want the pixel value to change when an image is flipped horizontally or vertically about that pixel, w will usually be symmetric about both the x and y axes. By similar logic, w should be diagonally symmetric so that 90° rotations in I will give 90° rotations in the digital image. In practice, the support of w will be only a few pixels across, so that $P(x_i, y_i)$ depends on values of I nearby (x_i, y_i).

If we'd like the overall radiance of the digital image to be similar to the overall radiance seen by the film plane, w should have unit volume:

$$\int w(x, y) dA = 1$$

If radiance scaling is desired, then some other constant than one can be used. This idea can be extended by requiring that the average intensity of the digital image is the same as the original continuous image. In other words, a small feature moving in the continuous image should not cause 'twinkling' in the digital image[36]. This can be stated quantitatively by requiring that the total contribution of a impulse (delta function) is the same regardless of position:

$$\sum_i \sum_j w(x - x_i, y - y_i) = \text{constant}$$

This constraint ensures that the DC component of the original and discrete images will be the same. This is equivalent to the constraint imposed by Mitchell and Netravali[74]. The example of an impulse in I also implies that w should be strictly nonnegative to avoid the possibility of negative pixel colors. In summary, we want w to have several features:

1. w has unit area.
2. w is horizontally, vertically, and diagonally symmetric.
3. The support of w has limited width.
4. $w \geq 0$ for all x and y.
5. $\sum_i \sum_j w(x - x_i, y - y_i) = \text{constant}$.

In addition, w may have either or both of two additional simplifying features[35]:

A. w is separable: $w(x, y) = a(x)a(y)$
B. w is rotationally symmetric: $w(x, y) = a(x^2 + y^2)$

A commonly used w that has all of the required features and is separable is a positive constant on a square centered at the origin. The width of the square is usually set to be the distance between pixel centers, but can be wider. This function is usually called a *box filter*.

In most of the graphics literature, the preceding discussion is usually viewed using signal processing theory. The image function I is convolved with a filter g, and becomes a new image function I'. The pixel values are then set by letting:

$$P(x_i, y_j) = I'(x_i, y_i) - (f \circ I)(x_i, y_i)$$

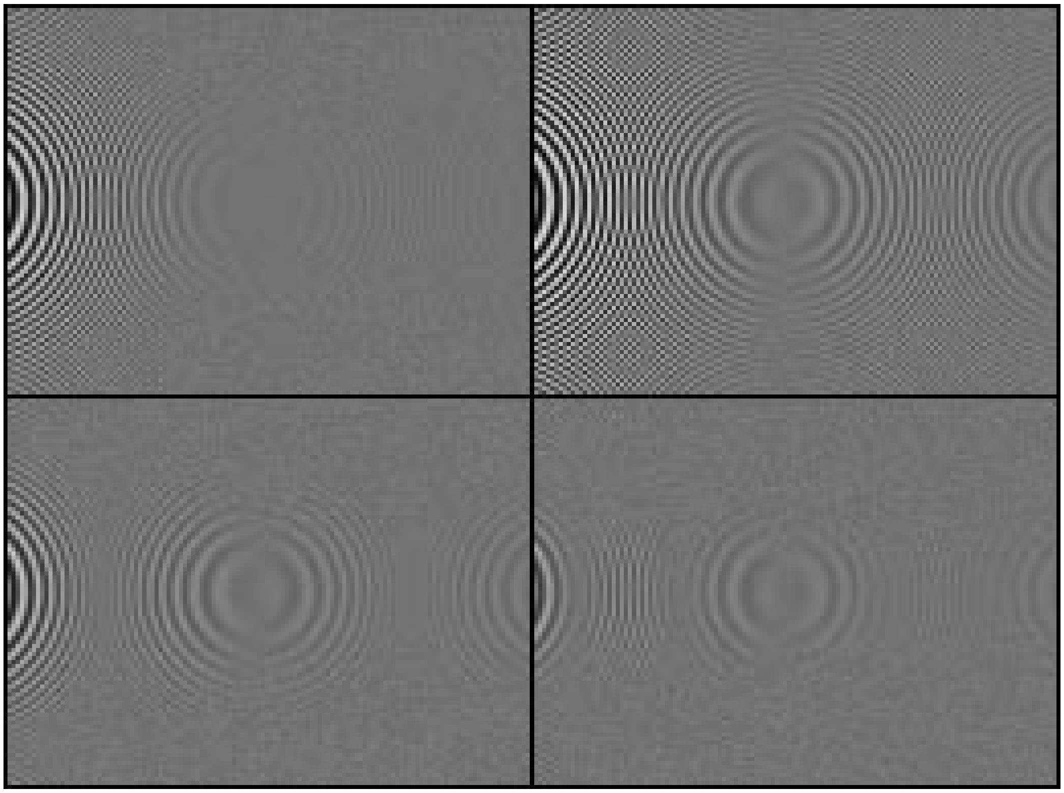

Figure 5.1: Images generated by (left to right, top to bottom) nonuniform filter, box filters with one pixel width, two pixel width, and three pixel width. The circular pattern on the left of each image is correct, while the circular pattern centered in the middle and on the right are caused by aliasing.

This approach is surveyed clearly by Blinn[13, 12]. Its limits and implications were deeply investigated by Kajiya and Ullner[59]. Because the shape of the intensity function of real pixels, and because our error metric is perceptual, signal processing theory does not yield an easy answer for what w is best[59]. There is, however, some consensus that signal processing theory implies that a nonuniform w with a maximum at the origin is preferable to a box filter[50, 51, 70, 36].

To develop an example of a nonuniform w, we can first assume a support that is restricted to at most a square of two pixel widths centered at the origin. We can further assume that w is separable ($w(x, y) = a(x)a(y)$) and that a is a cubic:

$$w(x, y) = \left(A|x|^3 + B|x|^2 + C|x| + D\right) \left(A|y|^3 + B|y|^2 + C|y| + D\right)$$

Note that this w is *not* circularly symmetric. Applying conditions 1-5 leaves yields four equations that imply $A = B = 0$, $C = -1$, and $D = 1$:

$$w(x, y) = (1 - |x|)(1 - |y|)$$

This w is similar to the bilinear filter shown in Figure 3 of [37]. In Figure 5.1, this nonuniform w and box filters of width one, two and three pixels is applied to the rather pathological image function $I(x, y) = \sin(x^2 + y^2)$. The origin is just to the left of each image. The concentric pattern on the left is 'real', and the others are artifacts caused by the regular grid of pixels and the character of w. The nonuniform w minimizes unwanted artifacts without excessive blurring of desired features.

It should be emphasized that the best w may be highly dependent on the display used. Amanatides and Mitchell have shown that NTSC displays in particular must be handled as a special case[4].

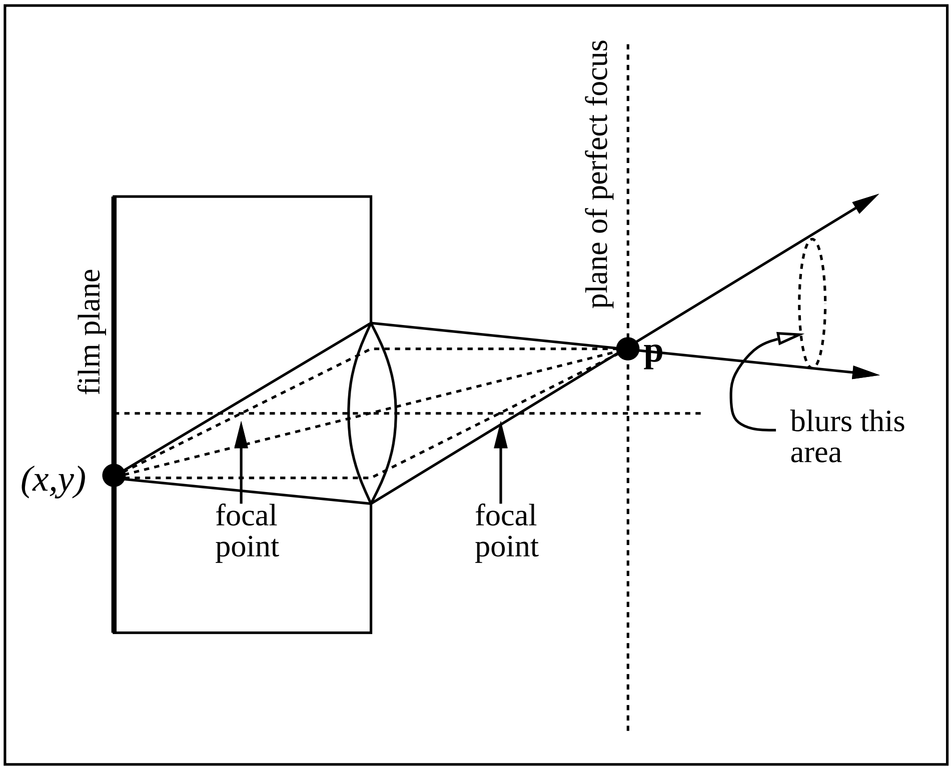

Figure 5.2: A thin lens camera

5.1.1 Finite Aperture Camera Model

A simple lens camera model can be substituted for a pin-hole camera model. This will make some objects appear to be blurred because of focusing effects. Such camera models have been used in both scanline[20, 86] and ray tracing[26] applications. In this model, the image function $I(x, y)$ is no longer the radiance seen through the pin-hole; instead it is the average radiance seen on the lens area from (x, y). The lens is assumed to be 'thin', so that it obeys certain rules illustrated in Figure 5.2: all light coming to point (x, y) passes through a point $\mathbf{p}$ on a plane of perfect focus; light traveling through the center of the lens will be undeflected; light passing through a focal point will be deflected by the lens along the axis of the lens. The second

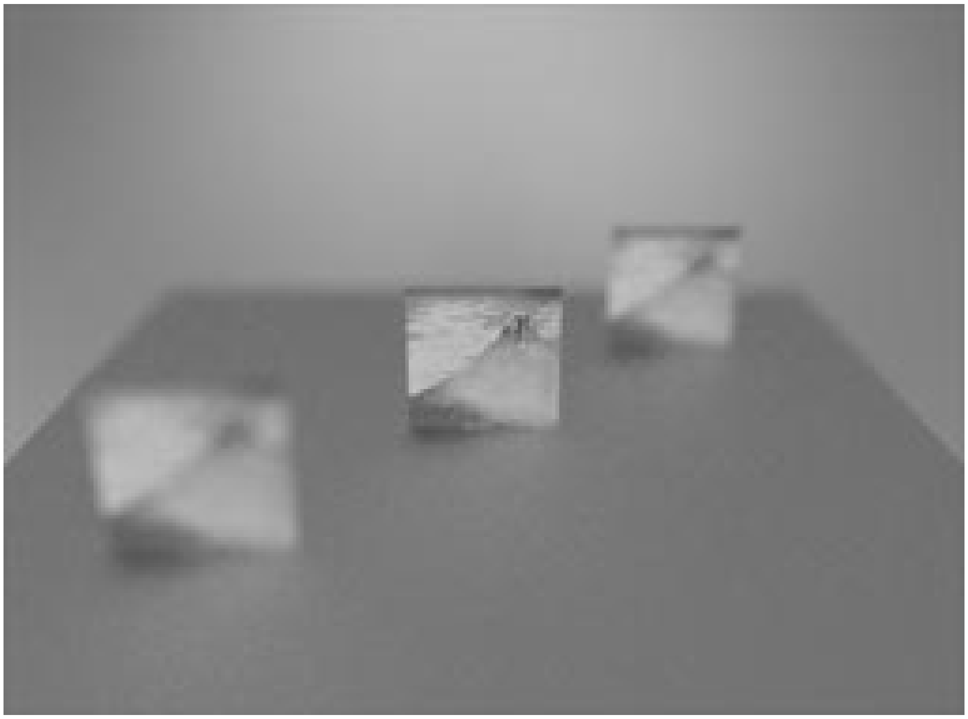

Figure 5.3: Three Fujis on brushed steel. The middle Fuji is in the plane of perfect focus.

and third rules can be used to determine $\mathbf{p}$ for a particular (x, y)[1]. Figure 5.3 shows a image calculated using a thin lens camera model.

Averaging the radiance seen at the lens means the image function for a lens of area A is:

$$I(x, y) = \frac{1}{A} \int_{\mathbf{q} \text{ on lens}} L(\mathbf{q}, \mathbf{q} - \mathbf{p}) dA$$

This means the expression for pixel intensity becomes:

$$P(x_i, y_j) = \frac{1}{A} \int \int_{\mathbf{q} \text{ on lens}} w(x - x_i, y - y_i) I(x, y) L(\mathbf{q}, \mathbf{q} - \mathbf{p}) dA dA'$$

Thus, even if L is known, creating a digital image for a particular viewpoint is not trivial! One thing to note is that I does not drop off as the solid angle subtended by the lens decreases when (x, y) strays from the center of the film (as is also true for the pin-hole model).

[1] This will break down for $(x, y) = (0, 0)$. Instead, the distance to the plane of perfect focus can be calculated, and the ray from (x, y) through the center of the lens will intersect the plane at $\mathbf{p}$.

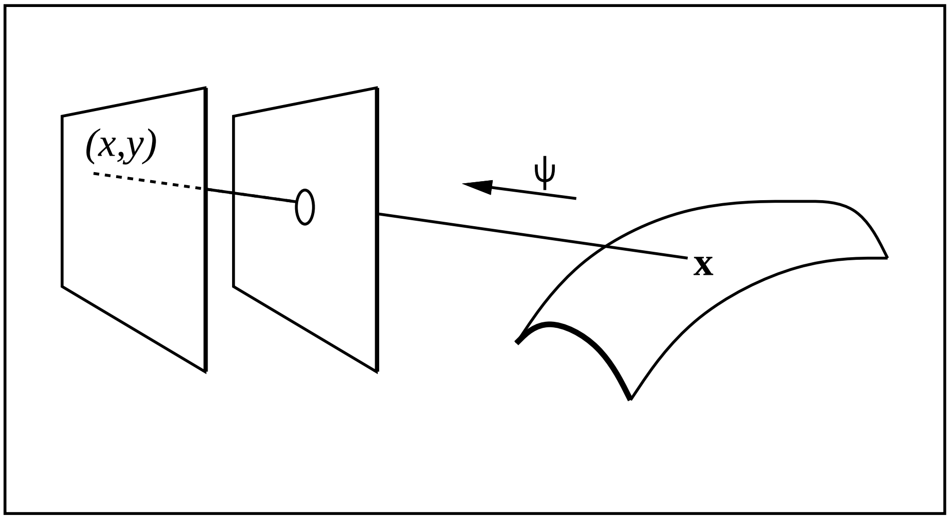

Figure 5.4: The image function at (x, y) is the radiance at $\mathbf{x}$ traveling toward the pin-hole.

5.2 Direct Lighting

As was seen in the last section, a pixel color can be determined by integrating the radiances seen in all directions. From the Ray Law, these radiance are the radiances L_{out} coming from the surfaces seen in those directions (see Section 3.2). For the pin-hole camera model, this means the image function I at a point (x, y) is:

$$P(x_i, y_j) = L_{out}(\mathbf{x}, \psi)$$

where $\mathbf{x}$ is the point on the surface seen through the pin-hole, and ψ is the direction from $\mathbf{x}$ to the pin-hole, as shown in Figure 5.4. To accurately find $L_{out}(\mathbf{x}, \psi)$, we would need to solve the rendering equation (Equation 3.4) at x. As an approximation, we can calculate the *direct light* reflected at x. Multiple reflections of light are not considered. Kajiya calls such methods *Utah Models* because of the pioneering work in this type of algorithm done at the University of Utah[61]. The benefit of a Utah Model is that the lighting calculation at $\mathbf{x}$ is entirely local.

Usually Utah Models assume that the light sources are point light sources infinitely far away. This allows Equation 3.4 to be evaluated for only one ψ_{in}.

5.2.1 Ambient Light

One problem with assuming only direct lighting is that the approximation is guaranteed to be too small. As a first approximation to fixing this problem, an arbitrary constant, the *ambient lighting*, is added to $L(\mathbf{x}, \psi_{in})$. The ambient term is supposed to approximate the *indirect* lighting at $\mathbf{x}$. Because indirect lighting is not constant, the ambient term will be in error for most $\mathbf{x}$. One technique for lowering the ambient error used in some graphics packages is to allow ambient terms to be specified for each object. Unfortunately, making good use of such a feature is more art than science.

One way to think of ambient lighting is to assume all radiance values visible from a point are some constant L_0. This L_0 is the appropriate value for the ambient component.

Researchers at Cornell have devised a method to intelligently guess a 'good' global ambient term for diffuse environments[22]. To do this they first calculate the average reflectance R and total surface area A in the environment. They then find the total power Φ emitted by all light sources. The fraction of Φ reflected immediately after being emitted by the sources is approximately $R\Phi$. Extending this idea to subsequent bounces estimates that the indirect power coming to a surface is approximately:

$$\Phi_{\text{indirect}} \approx \Phi(R + R^2 + R^3 + \cdots) = \Phi \frac{R}{1 - R}$$

Using Equation 4.1, this implies the ambient light reflected at $\mathbf{x}$, $L_a(\mathbf{x})$, is:

$$L_a(\mathbf{x}) = R(\mathbf{x}) \frac{\Phi_{\text{indirect}}}{\pi A}$$

5.3 Whitted-Style Ray Tracing

The Utah Models perform best for primarily diffuse scenes. Kay used Snell's Law and ray tracing to include refractive effects[64]. Whitted used slightly more general ray tracing techniques to extend Utah Models to include perfect specular effects and shadows[121]. His technique is usually called *ray tracing*, but because that term has become so overloaded, I will refer to it as *Whitted-style* ray tracing.

In Whitted-style ray tracing, the image function $I(x, y)$ is calculated by sending a ray from (x, y) through the pin-hole, and determining the first point $\mathbf{p}$ hit by the ray. If $\mathbf{p}$ lies on a non-specular surface, then a Utah model is applied. However, the contribution of a particular light source is only counted if $\mathbf{p}$ is not in shadow relative to that source. Whether a point is in shadow is determined by sending a ray toward the light source and seeing if it hits any objects before the light.

If $\mathbf{p}$ is on a specular surface, then the radiance is calculated by attenuating the radiance seen in the direction of reflection. If the surface is not opaque, the attenuated color in the transmitted direction is added. Figure 5.5 shows how Whitted's method would process several rays.

We can implement Whitted's approach as a recursive function that evaluates the Global Radiance Function for a particular viewpoint. Suppose we have such a function, $L_p(x, y)$ defined for the film plane. This function could be written in terms of the Global Radiance Function $L(\mathbf{p}, \psi)$:

```
radiance function $L_p$(real $x$, real $y$)
        *$L_p$ returns the radiance value seen at $(x, y)$ on film plane*
        *coming from direction of pin-hole.*
**begin**
        direction $\psi$
        point $\mathbf{o}$
```

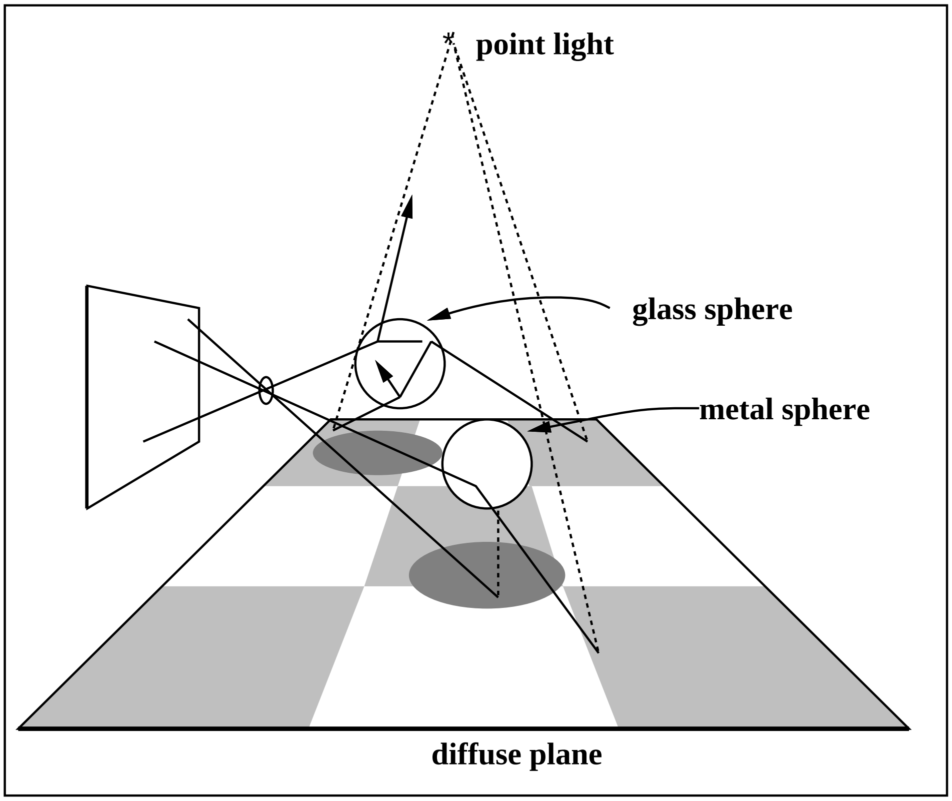

Figure 5.5: Several rays traced from the film plane. The solid lines are viewing rays, and the dashed lines are shadow rays.

```
        point $\mathbf{ph}$
        $\mathbf{o}$ = position of $(x, y)$ in object space
        $\mathbf{ph}$ = position of pin-hole in object space
        $\psi = \mathbf{o} - \mathbf{ph}$
        return $L(\mathbf{o}, \psi)$
    end ($L_p$)
```

The Global Radiance Function also returns a radiance:

```
    radiance function $L$(point $\mathbf{o}$, $\psi$)
        L returns the radiance value seen at o coming from direction $\psi$
    begin
        point $\mathbf{p}$
        if ray $\mathbf{o} - t\psi$ misses everything then
            return background radiance
        else
            find intersection point $\mathbf{p}$ of first object hit by ray
            if (object is opaque specular)
                find incoming reflected direction, $\psi_r$, by Equation 2.2
                return $k_s L_{in}(\mathbf{p}, \psi_r)$
            else if object is transparent specular
                find incoming reflected direction, $\psi_r$,by Equation 2.2
                find incoming transmitted direction, $\psi_t$, by Equation 2.3.
                return $k_s L_{in}(\mathbf{p}, \psi_r) + k_t L_{in}(\mathbf{p}, \psi_t)$
            else apply Utah Model
                $\psi_s = (\mathbf{l} - \mathbf{p})$
                if ray $\mathbf{p} - t\psi_l$ hits something close than $\mathbf{l}$ then
                    return $R(\mathbf{p})L_{ambient}$
                else
                    return $R(\mathbf{p})L_{ambient}$+ direct lighting from $\mathbf{l}$.
    end ($L_p$)
```

If a series of specular objects is hit we will have infinite recursion. Whitted avoids this by returning zero radiance after a certain number of reflections. Hall and Greenberg suggest stopping the recursion adaptively based on accumulated attenuation[44]. The adaptive technique is especially good when clear objects are present, and the internal reflections cause branching.

In some sense, Whitted-style ray tracing simply provides Utah-shaded objects, and reflections of Utah-shaded objects. This is coupled with the shadow ray technique that allows objects to shadow one another.

5.4 Stochastic Ray Tracing

Among the problems with Whitted-style ray tracing, and most other techniques that preceded ray tracing, is that they do not account for a finite aperture camera, non-point light sources, area sampling of I, or non-specular indirect lighting. Cook et al. attacked all of these problems at once by realizing that the intensity level for a pixel can be written as a multidimensional integral, and that classic Monte Carlo integration techniques can be used to solve that integral[26]. In this section Cook et al.'s technique is presented, followed by the other stochastic techniques of Kajiya and Ward et al.

5.4.1 Cook et al.'s Distributed Ray Tracing

The fundamental idea of Cook et al. is to perform a numerical integration for every pixel[26, 25, 14, 41]. Rather than using conventional regular quadrature techniques, they use stochastically distributed sample points. Using random sample points does not necessarily have a smaller error than regular sampling[2], but the random method's error will be less visually objectionable because there will not be coherence in the error between pixels.

For some insight into how the numerical integration is applied, consider the expression for the intensity of a pixel:

$$P(x_i, y_j) = \int\int w(x - x_i, y - y_i) I(x, y) dx dy \tag{5.1}$$

In Appendix B, it is shown that we can approximate an integral with a *primary unbiased estimator*:

$$\int_{a' \in \Omega} h(a') d\mu(a') \approx \frac{h(a)}{f(a)}$$

[2] Traditionally, Monte Carlo integration has better asymptotic error behavior if the dimension of the integral is sufficiently large[109], as it often is in graphics applications.

where a is a random variable with probability density function f. Saying that $h(a)/f(a)$ is an unbiased estimator for the integral simply means that the expected value is the value of the integral. We can come up with a 'better' unbiased estimator for the integral by averaging many of the primary estimators to form a *secondary estimator*:

$$\int_{a' \in \Omega} h(a') d\mu(a') \approx \frac{1}{N} \sum_{i=1}^{N} \frac{h(a_i)}{f(a_i)}$$

The secondary estimator is better simply because it has a lower variance. The Law of Large Numbers tells us that the secondary estimate will converge to the value of the integral with probability one as N goes to infinity.

Assuming we know how to evaluate I, we can easily write down a primary estimator for the integral of Equation 5.1. First, assume that the pixel area is one (the distance between pixel centers is one), and that w is zero outside the pixel area. Using a constant probability density function $f = 1$ inside the pixel, and $f = 0$ outside the pixel will generate random points a on the pixel area. Thus the primary estimator will be:

$$P(x_i, y_j) = \int \int w(x - x_i, y - y_i) I(x, y) dx dy \approx I(a_x, a_y) \tag{5.2}$$

As discussed earlier, the secondary estimator is found by averaging a series of the primary estimators. Stratified sampling can be employed by subdividing the domain of the integral in Equation 5.2 and summing the primary estimator of each of these integrals.

Suppose instead that we use the nonuniform $w = (1 - |x|)(1 - |y|)$ with a width of two. Without loss of generality, assume that $x_i = y_j = 0$ (a change of coordinates):

$$P(x_i, y_j) = \int_{-1}^{1} \int_{-1}^{1} (1 - |x|)(1 - |y|) I(x, y) dx dy \tag{5.3}$$

A naive primary estimator can again be found with a uniform density $f(a_x, a_y) = 0.25$ on the support of w:

$$\int_{-1}^{1}\int_{-1}^{1}(1-|x|)(1-|y|)I(x,y)dxdy \approx 4(1-|a_x|)(1-|a_y|)I(a_x,a_y) \tag{5.4}$$

We can instead use a nonuniform f for choosing sample points with density f. If our choice of f is wise (i. e. reduces variance of the primary estimator), then we are using *importance sampling*. A natural choice is $f = w$ because the expressions are simplified:

$$\int_{-1}^{1}\int_{-1}^{1}(1-|x|)(1-|y|)I(x,y)dxdy \approx I(a_x,a_y) \tag{5.5}$$

Using a nonuniform f will require generating nonuniform random numbers. As shown in Appendix B, a series of one dimensional independent identically distributed according to f random variables $(\alpha_1, \alpha_2, \alpha_3, \ldots)$ (abbreviated $\alpha_i \sim f$) can be generated by suitably transforming a series of *canonical* random numbers $(\xi_1, \xi_2, \xi_3, \ldots)$. Canonical random numbers are simply uniformly distributed random numbers between zero and one. The actual transformation for a given f is:

$$\alpha_i = F^{-1}(\xi_i) \tag{5.6}$$

where F^{-1} is the inverse of the probability distribution F associated with the probability density f:

$$F(x) = \int_{-\infty}^{x} f(x')dx' \tag{5.7}$$

Generating multidimensional random variables is more difficult, but can usually be done in a generalization of the inverse distribution procedure if f is sufficiently well behaved (see Appendix B). For separable densities, $f(x,y) = g(x)h(y)$, we can choose (α, β) pairs with density f by choosing α according to g and β according to h.

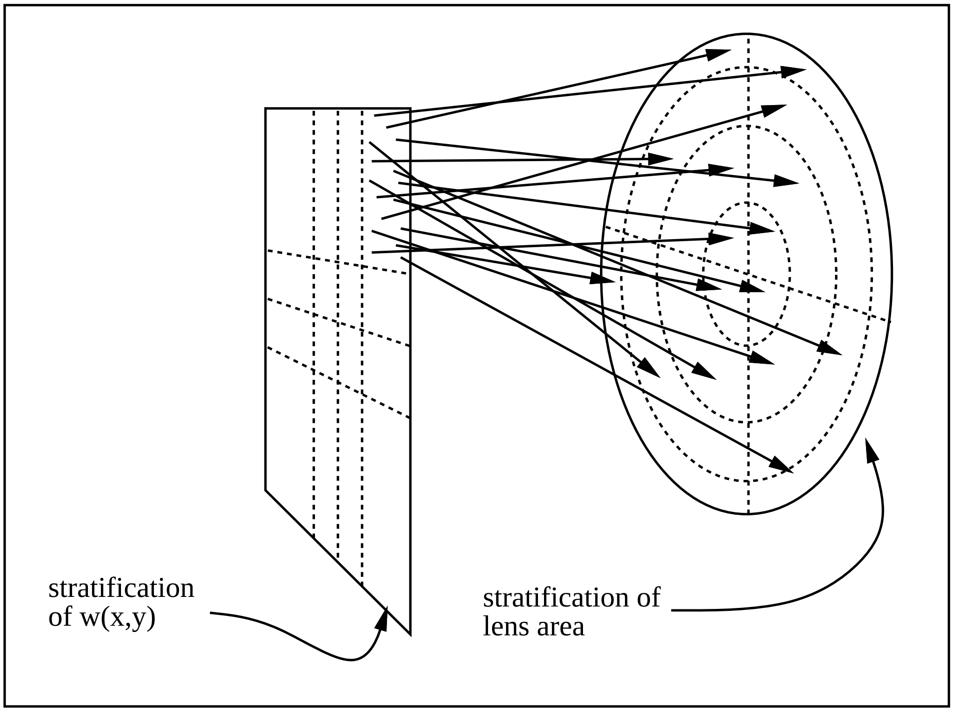

Figure 5.6: 16 rays fired from one stratum of pixel toward 16 strata on lens

.

Getting back to our pixel sampling, the weighting function $f(x, y) = (1 - |x|)(1 - |y|)$ is separable, and $g = h$, so we can generate (α_x, α_y) pairs according to $f(x) = (1 - |x|)$. The distribution function for this f is:

$$F(x) = \int_{-1}^{x} (1 - |x'|)dx' = \frac{1}{2} + x - \frac{1}{2}x|x| \tag{5.8}$$

and thus the inverse of F is:

$$F^{-1}(x) = \begin{cases} 1 - \sqrt{2(1-x)} & \text{if } x \geq 0.5 \\ -1 + \sqrt{2x} & \text{if } x < 0.5 \end{cases} \tag{5.9}$$

Thus, using $(F^{-1}(\xi_i), F^{-1}(\xi_j))$ from Equation 5.9 will generate pairs (α_i, α_j) with density $f(x, y) = (1 - |x|)(1 - |y|)$.

An immediate thing to wonder is whether we can mix importance sampling and stratified sampling. This actually can be done in a very simple manner: pick a set of stratified canonical

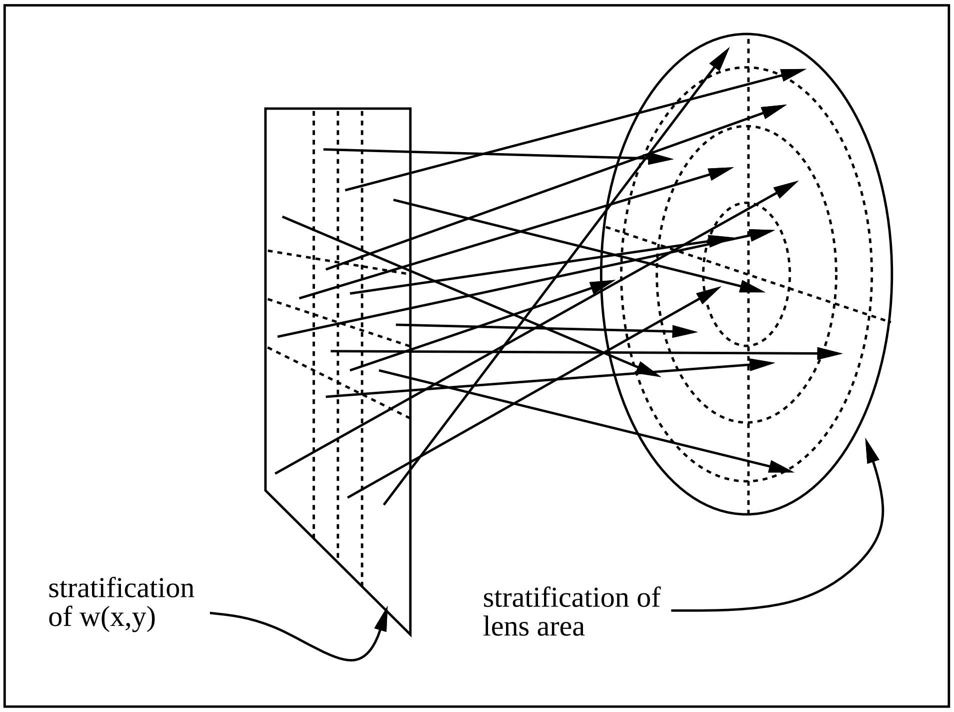

Figure 5.7: 16 rays fired from all strata of pixel toward 16 strata on lens

.

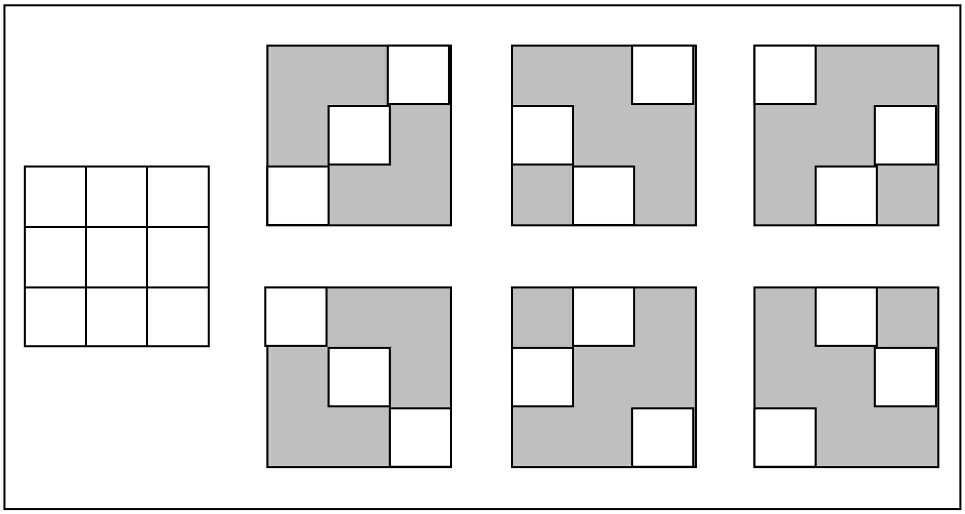

Figure 5.8: All valid permutations of three uncorrelated strata

.

samples (ξ_i, ξ_j) from the unit square, and transform them using the inverse distribution function. We are actually sampling according to a different probability function in each stratum, but we can add the estimators as if they were identically distributed. This idea *greatly* simplifies implementing a stochastic ray tracing code.

The pixel area is not the only space that needs to be integrated over. If we add a camera lens model, then we have a four dimensional integral. If we put a polar coordinate system on the camera lens we have (for w with support of width 2):

$$P(x_i, y_j) = \frac{1}{A_{lens}} \int_{x=-1}^{1} \int_{y=-1}^{1} \int_{\theta=0}^{2\pi} \int_{r=0}^{R_{lens}} w(x,y) L(x,y,r,\theta) r \sin\theta \, dr \, d\theta \, dx \, dy$$

To straightforwardly apply Monte Carlo integration we would generate four dimensional random variables to generate primary estimators. Figure 5.6 shows a pixel sampling function w and lens area each divided into 16 strata. This makes $16^2 = 256$ strata in total, so 256 rays will be fired in all. In the figure, the sixteen rays that would come from one of the pixel stratum is shown. All of the other strata on the pixel would also send this bundle of rays. As the dimension of the integral grows larger (as it will once we add shadows and reflection and motion blur), the explosion of rays will increase, so the number of strata for each dimension must be cut to keep the number of rays at a reasonable level. This problem was avoided by Cook using what he called uncorrelated jittering[25]. In this method, we associate each stratum on the pixel with a stratum on the lens, and make sure no stratum has more than one association. In this way, one ray is fired through each stratum, as shown in Figure 5.7.

Cook uses the term *uncorrelated* because any consistent mapping between particular strata will cause artifacts in the image. Instead we should use a different mapping for each pixel. Uncorrelated jittering is especially helpful when some of the dimensions of the integral are constant. For example, if the surface seen from the pixel is in perfect focus (the same point is seen regardless of the point chosen on the lens), then we still get a full stratification of the pixel space.

One problem with uncorrelated jittering is that the mathematics behind it has never been investigated in the Computer Graphics literature. This lack of foundation can cause confusion when trying to extend the technique. The basis for uncorrelated jittering can best be seen by looking at a two dimensional example. Suppose we have the unit square divided into 9 equal squares. As discussed earlier, we can get a primary estimator for an integral over the square by evaluating an expression at a random point within the square. Another way to get a primary estimator would be to choose one of the 9 squares at random and then choose a random point within the square. We could extend this idea by selecting more than one square and taking one sample from each square. This will still be an unbiased estimator as long as each square is equally likely to be chosen in the long run. In two dimensional uncorrelated jittering, we would choose 3 of the 9 squares, making sure that each square is the sole occupant of each row and column. This will allow full stratification in both dimensions. All such sets of three squares are shown in Figure 5.8. If we choose any one of these allowed sets of three squares at random, the estimator will be unbiased because each square is a member of the same number of allowed sets. One way to generate the sets is to permute the sequence $(1, 2, 3)$ and use this as row numbers and the unpermuted $(1, 2, 3)$ as column numbers.

This basic idea of uncorrelated sampling is relatively unknown in the Monte Carlo literature. It is briefly mentioned as an untried possibility in the book by Kalos[63], but Computer Graphics seems to be the only field in which it has been applied. The specifics of uncorrelated jittering add some complexity to the basic idea; the two pixel dimensions (x and y) and the two lens dimensions are each linked as a pair. Valid mappings between pixel strata $(p_1, p_2, \cdots, p_N)$

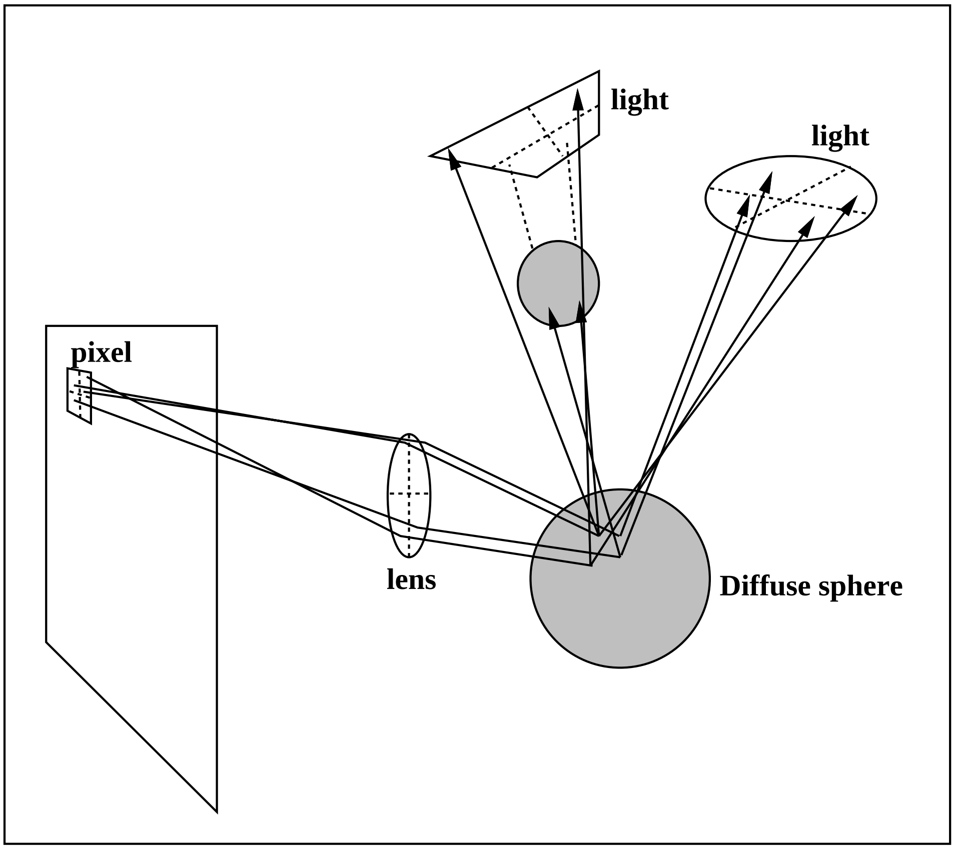

Figure 5.9: Rays traced in distributed ray tracing with four rays per pixel.

and lens strata $(l_1, l_2, \cdots, l_N)$ can still be found by permuting one of the sequences. Cook actually used a more restrictive mapping method that avoided mappings that allowed too much correlation[3]. This would avoid sample sets that had many rays in the same area of the pixel going to the same area of the lens. This is no longer a strictly Monte Carlo procedure, but can be justified if every stratum gets 'equal opportunity'. This technique of restricting the acceptable random sets relates to traditional 'quasi-random' methods.

One nice thing about uncorrelated jittering is that extra dimensions can be easily added. For example, if a diffuse reflector is seen from a particular pixel, we'll want to do a lighting calculation. This will be in the form of an integral across each light source area. Using

[3] It is not clear from Cook's presentation that his strategy yields an unbiased estimator.

Equation 3.5 yields:

$$P(x_i, y_j) = \frac{1}{A_{lens}} \int_{x=-1}^{1} \int_{y=-1}^{1} \int_{\theta=0}^{2\pi} \int_{r=0}^{R_{lens}} \int_{x' \text{ on light}} w(x, y) g(\mathbf{x}, \mathbf{x}') R(\mathbf{x}\lambda) I(\mathbf{x}') \, dr \, d\theta \, dx \, dy$$

Here $\mathbf{x}$ is the point seen by a particular ray from the lens and $g(\mathbf{x}, \mathbf{x}')$ is evaluated by sending a shadow ray toward the light. $I(\mathbf{x}')$ represents the lighting expression from Equation 3.5. The rays traced for this situation are shown in Figure 5.9. If the object hit is not diffuse then instead of testing for shadow and shading, a reflected ray is sent.

5.4.2 Kajiya's Path Tracing

Kajiya extended distributed ray tracing by phrasing the problem as a Monte Carlo solution to an integral equation. Recall that the radiance at a point was written down in Equation 3.5, which can be written without wavelength dependency as:

$$L_{out}(\mathbf{x}, \psi) = L_e(\mathbf{x}, \psi) + \int_{\text{all } \mathbf{x}'} g(\mathbf{x}, \mathbf{x}') \rho(\mathbf{x}, \psi, \psi') L_{out}(\mathbf{x}', \psi') \cos\theta \frac{dA' cos\theta'}{\|x' - x\|^2} \tag{5.10}$$

In Appendix B it is shown that for an equation of the form:

$$a(x) = b(x) + \int_{x' \in \Omega} k(x, x') a(x') d\mu(x') \tag{5.11}$$

We can write down an unbiased primary estimator:

$$\begin{aligned} a(x) \;=\; & b(x) + \\ & \frac{k(x, x^1) b(x^1)}{f_1(x^1)} + \\ & \frac{k(x, x^1) k(x^1 x^2) b(x^2)}{f_2(x^1, x^2)} + \\ & \frac{k(x, x^1) k(x^1, x^2) k(x^2, x^3) b(x^3)}{f_3(x^1, x^2, x^3)} + \\ & \vdots \\ & \frac{k(x, x^1) \cdots k(x^{n-1}, x^n) b(x^n)}{f_n(x^1, \cdots, x^n)} + \end{aligned}$$

$$\vdots \tag{5.12}$$

where $f_n(x^1, \cdots, x^n)$ is the probability density function for a sequence $(x^1, \cdots, x^n)$. The series can be terminated by waiting until we get a k with value zero (or accepting truncation error), or by *russian roulette*, where it is terminated probabilistically[6]. Russian roulette eliminates the bias of truncation.

To directly apply this series method to the rendering equation, we can view the space Ω as $\mathbf{x}^\star$, the set of all points on all surfaces. Once we have chosen a point on the pixel and lens, then we want to know $L(\mathbf{x}_0, \psi)$, where $\mathbf{x}_0$ is a point on the lens, and ψ is an incoming direction determined by the thin lens rules. We trace a ray to find the first surface $\mathbf{x}_1$ seen in direction $-\psi$. By the Ray Law $(\mathbf{x}_0, \psi) = L_{out}(\mathbf{x}, \psi)$. Equation 5.10 gives an expression for $L_{out}(\mathbf{x}, \psi)$ in the form of Equation 5.11. The function L_{out} maps to a, L_e maps to b, and the complicated expression in the integrand (with L divided out) maps to k.

To get an estimator we just need to choose a series of points according to some distributions f_n, and evaluate the series. For each series of n terms, we need to do $(n-1)$ evaluations of the visibility term g for adjacent points. This is accomplished by tracing a ray between the points and seeing if they are visible to each other. In practice, we should choose the points carefully. Kajiya suggests eliminating zero terms by only using series that have visible adjacent pairs. If we carry this idea farther by setting the probability functions to be proportional to the k (automatically setting zero probability for zero terms) and allow truncation we get:

$$\begin{aligned}
a(x) \quad \approx \quad & b(x) + \\
& K(x, x^1) b(x^1) + \\
& K(x, x^1) K(x^1 x^2) b(x^2) +
\end{aligned}$$

$$
\begin{aligned}
&K(x, x^1)K(x^1, x^2)K(x^2, x^3)b(x^3) + \\
&\vdots \\
&K(x, x^1) \cdots K(x^{n-1}, x^n)b(x^n) \qquad (5.13)
\end{aligned}
$$

where $K(x^{n-1}, x^n)$ is the volume of $k(x^{n-1}, x^n)$. Applying this to the rendering equation gives:

$$
\begin{aligned}
L_{out}(\mathbf{x}, \psi) \quad \approx \quad & L_e(\mathbf{x}, \psi) + \\
& R(\mathbf{x}, \psi)L_e(\mathbf{x}', \psi') + \\
& R(\mathbf{x}, \psi)R(\mathbf{x}', \psi')L_e(\mathbf{x}'', \psi'') + \\
& \vdots \qquad (5.14)
\end{aligned}
$$

Where x^n is chosen by sending a ray from $x^{(n-1)}$ in direction $-\psi^{(n-1)}$, and $\psi^{(n-1)}$ is chosen with a density given by the **SPF** at $\mathbf{x}^{n-1}$ given incoming direction $\psi^{(n-2)}$. This *sounds* very complicated, but in practice it's very simple. Starting on the lens, send a ray and find the point hit. If the point emits any light, accumulate it. Reflect the ray as if it were a real light ray traveling 'forward' (this is allowed because of the Helmholtz reciprocity condition), and find the new surface hit. If this second point emits light accumulate its value times the reflectivity of the first surface. Send a new ray according the the **SPF** of the second surface and find the third surface hit. If the third surface emits light, accumulate its value times the product of the reflectivities of the first two surfaces. The third surface sends a reflected ray, and so on. The process stops when the product of the reflectivities falls below a certain value, or at an arbitrary number of reflections. Since the method traces light paths (in reverse) through the room, Kajiya called it *path tracing.* An example of this process is shown in Figure 5.10, where a room with uniform diffuse reflectivity 0.5, except for a glass ball and a light source with radiance 8 and zero reflectivity. The series is terminated when the accumulated reflectivity falls

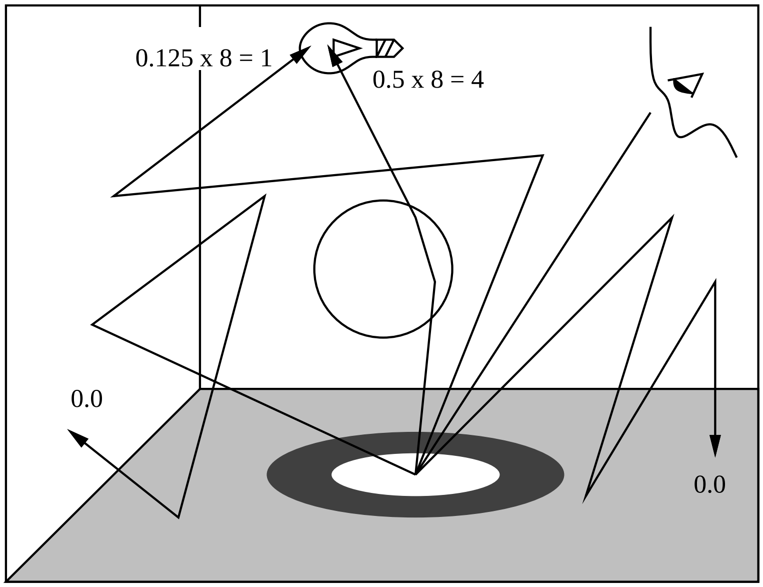

Figure 5.10: Path tracing in a room with a glass ball, walls with reflectivity 0.5, and a light with radiance 8.

below 0.1 (4 bounces). In the four rays shown, two take four reflections and contribute nothing. Two others contribute 1 and 4. The estimate for radiance is the average of these, or 1.25.

The problem with this technique is that the variance will be very high unless the emitted light is divided over a large area. Kajiya tried to lessen this problem by calculating the direct lighting at each selected point. This is best thought of as recursively applying distributed ray tracing. When a diffuse surface is seen, a reflection ray is sent in addition to the shadow ray. If the reflection ray *directly* hits a light source, the direct contribution is not included. If it hits the light source after reflection from a specular surface, then the contribution is counted. These indirect contributions allow for effects like the bright spot under the glass ball in Figure 5.10. Because these indirect terms are handled in the same way as those of crude path tracing, the indirect lighting may have very high variance.

Kajiya's path tracing can be thought of as lazy evaluation of the global radiance function at the lens; only those radiances at points contributing to the image are calculated. Unfortunately, many surfaces that are not seen will contribute their radiances indirectly, and the radiances of these surfaces might be recalculated many times. Ward et al. suggested that once a radiance value is calculated it should be saved in a geometric table for later use[119]. This type of storage was implemented for diffuse reflectors, where the table needed no directional information. Ward et al.'s technique is especially effective for diffuse interreflection. Like path tracing, it has fairly high variance for effects like bright spots under glass balls.

5.4.3 Shadow Ray Optimization

One problem with traditional ray tracing methods is that shadow rays are sent toward every light source[121, 25]. An example of why this is a problem would occur when ray tracing an image of a street lit by one hundred streetlights. At any particular spot on the street, we will send one hundred shadow rays in total, even though most of the lights make negligible contributions. Kajiya noted that shadow rays could be sent in various numbers in a probabilistic way, but did not propose a specific strategy. Shirley implemented a method where one shadow ray is sent toward all light sources, and the contribution is either all or nothing depending on whether the ray is obstructed[100].

The difficult part of sending the shadow ray is constructing the probability space that determines where the ray is sent. The first step is to choose the target light based on its total contribution. This way more attention is paid to the most important lights (the nearby streetlights in the earlier example). Then a point on the light source is chosen to send the shadow ray toward. This should all be done using Cook's uncorrelated jittering, otherwise all of the rays through a pixel could go to the same area on the same light. Given a set of canonical random number pairs $(\xi_1, \xi_2)_i$ chosen in a stratified manner from the unit square, we choose

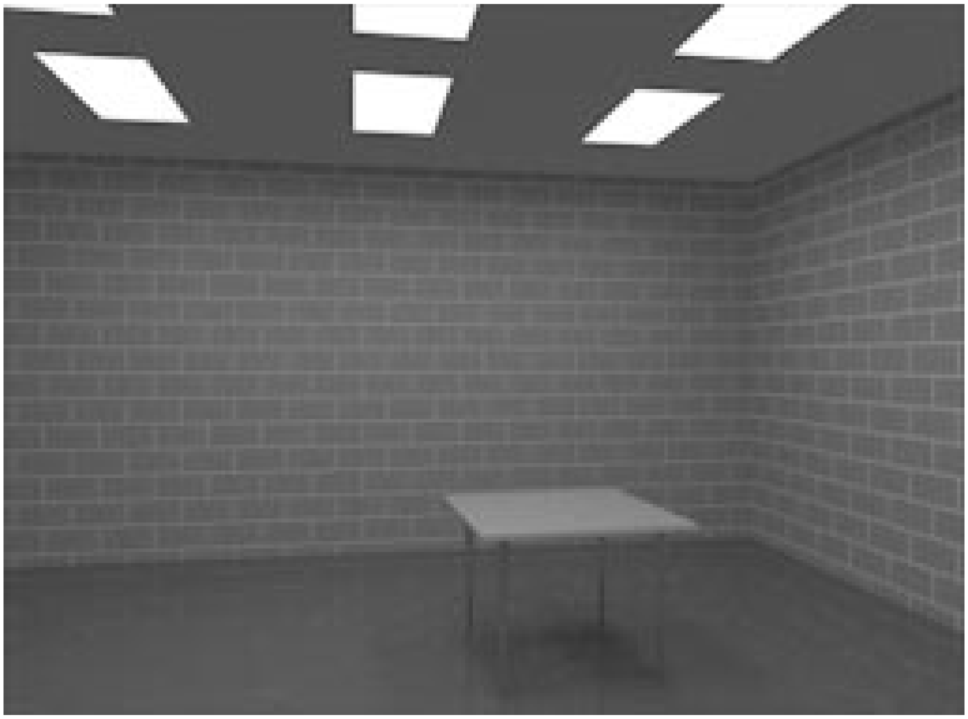

Figure 5.11: Room with one shadow ray per viewing ray.

the target light by using just ξ_i. Suppose we have two lights that contribute radiances of 1 and 9 at the target point. If $\xi_1 < 0.1$, then the first light will receive the shadow ray. Otherwise the second will get it. If the first light receives the ray ($\xi_1 < 0.1$), then we know that $(10\xi_1, \xi_2)$ are a pair of canonical random numbers, and this pair is used to choose a spot on the target light. This idea can be generalized to N lights by setting up N intervals for ξ_1 and dividing by the width of the chosen interval.

Figure 5.11 shows a room lit with nine lights. Each of the 16 viewing rays produced only one shadow ray. Figure 5.12 shows an art gallery lit by 5 spotlights with one shadow ray for each of the sixteen viewing rays per pixel. Since the spotlights are very directional, most locations send almost all shadow rays to one light. Both figures have indirect lighting calculated by the zonal techniques discussed in the next chapter.

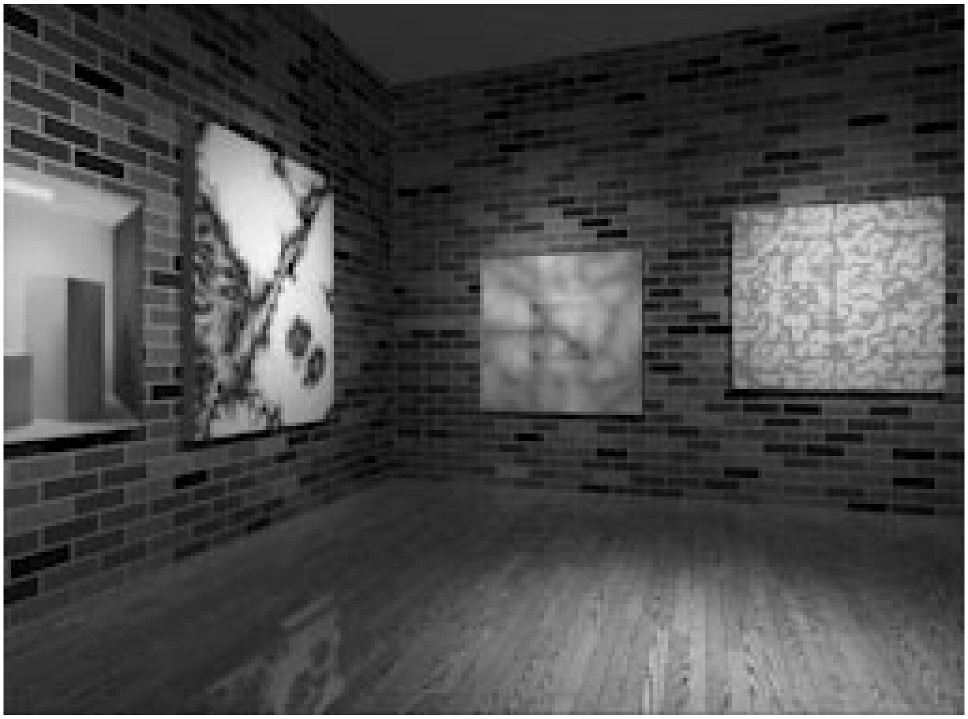

Figure 5.12: Room with one shadow ray per viewing ray.

5.5 Strategies for Non-adaptive Sampling

In the last section, the integrals used to calculate pixel brightness were phrased in terms of stratified Monte Carlo sampling. This sampling occurred on the two dimensional spaces of pixel area, lens area, light source area, and reflection ray direction. Each of these two dimensional spaces was sampled using uncorrelated jittering, where each of the spaces was fully stratified, and the stratum of each space was paired with a stratum of each other space in an irregular manner.

A crucial part of this sampling process is intelligently selecting the sampling points. The easiest way to select sample points is to set a predetermined number of samples for each pixel, and select a pattern of that many points in a two dimensional probability space for each of the

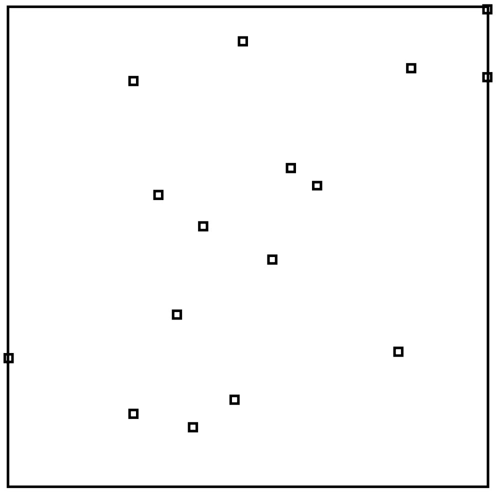

Figure 5.13: 16 random sample points

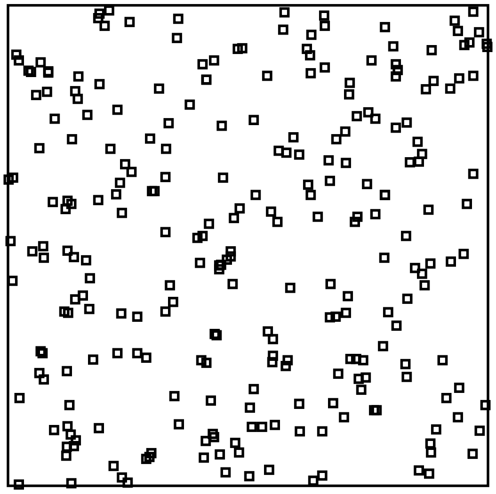

Figure 5.14: 256 random sample points

two dimensional spaces of the integral. This section examines how to best arrange these points once their number is known. The next section discussed adaptive methods.

5.5.1 Random Sampling

The simplest way to choose N points (x_i, y_i) from the 'canonical' two dimensional probability space is to pick every x_i and y_i independently by setting them to canonical pairs (ξ_i, ξ'_i). A set of 16 random pairs is shown in Figure 5.13, and a set of 256 random pairs is shown in

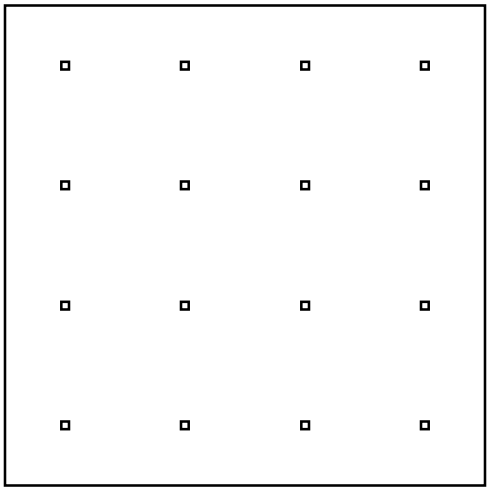

Figure 5.15: 16 regular sample points

Figure 5.14. As can be seen in these figures, random sampling allows some areas to be sparsely sampled, and others to be densely sampled. This implies that the variance in estimates using random sampling will be high.

5.5.2 Regular Sampling

We could simply place the pairs evenly in a grid pattern (as is done in traditional quadrature methods), as shown in Figure 5.15. This will prevent clumping of samples, but may introduce spatial regularity into the error across many pixels.

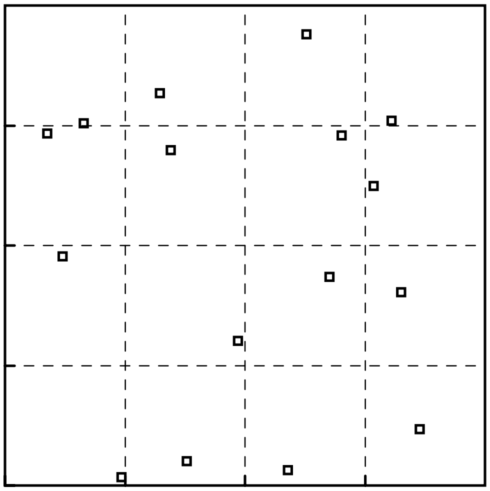

Figure 5.16: 16 jittered sample points with one sample per square

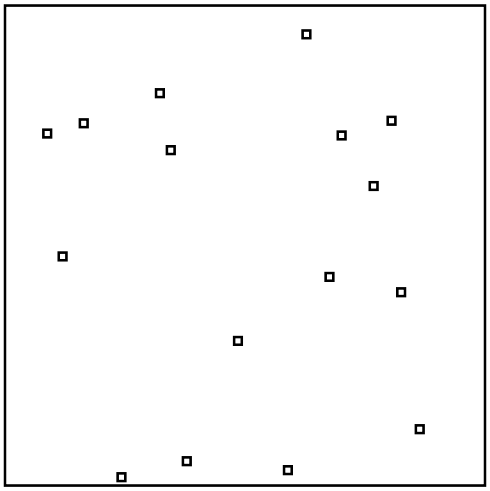

Figure 5.17: 16 jittered sample points

5.5.3 Jittered Sampling

Jittered sampling is another name for classical Monte Carlo stratified sampling[26]. Typically, the canonical probability space is partitioned into an n by n set of equal area squares, and a point is chosen uniformly from each square. A jittered pattern of 16 samples is shown in Figure 5.16, and without the square boundaries in Figure 5.17. A pattern of 256 points is shown in Figure 5.18.

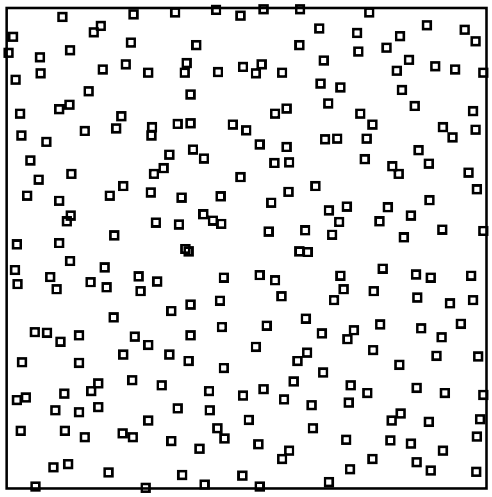

Figure 5.18: 256 jittered sample points

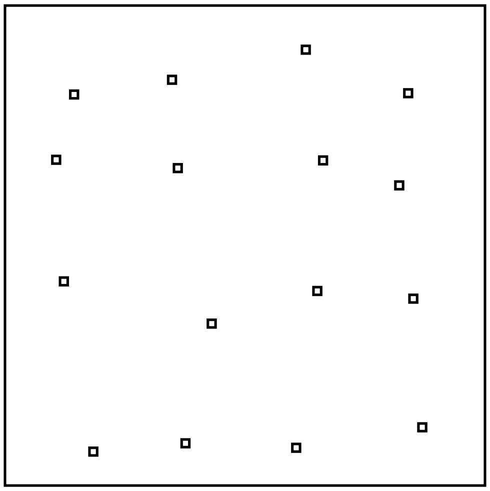

Figure 5.19: 16 semijittered (0.5 maximum shift) sample points

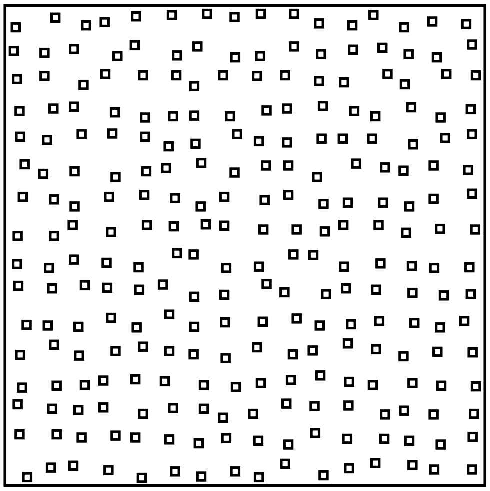

Figure 5.20: 256 semijittered (0.5 maximum shift) sample points

One problem with jittering is that limited clumping can take place. This problem can be lessened by choosing points nearer to the center of each square. Cook did this by using a Gaussian distribution to choose from each square. A simpler version of this is shown in Figure 5.19, where the samples are *half-jittered*: points are chosen uniformly within the square, half the width of the full square. A set of 256 half-jittered sample points is shown in Figure 5.20. As can be seen, there is less clumping, but more regularity than with simple jittering.

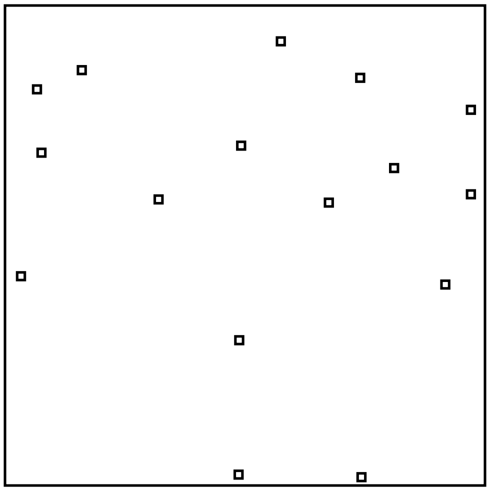

Figure 5.21: 16 Poisson disk sample points (minimum separation 0.1 pixel width)

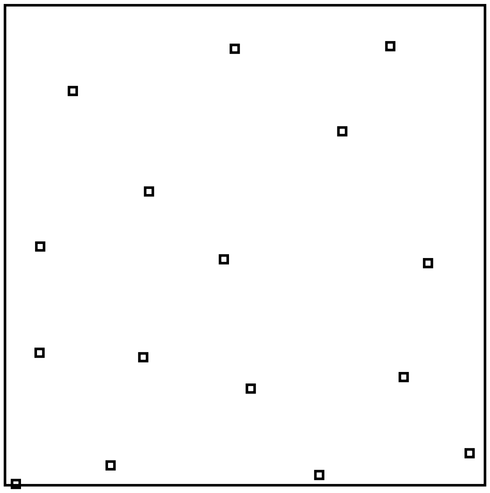

Figure 5.22: 16 Poisson disk sample points (minimum separation 0.2 pixel width)

5.5.4 Poisson Disk Sampling

A simple way to avoid clumping of sample points is to generate a sequence of samples, and reject a new sample if it is too close to an existing sample. This method, called Poisson disk sampling, has often been used in ray tracing applications for this reason[30, 25]. A set of 16 samples with minimum separation 0.1 is shown in Figure 5.21, and with minimum separation 0.2 in Figure 5.22. Poisson disk distributions of 256 samples are shown in Figures 5.23 and 5.24.

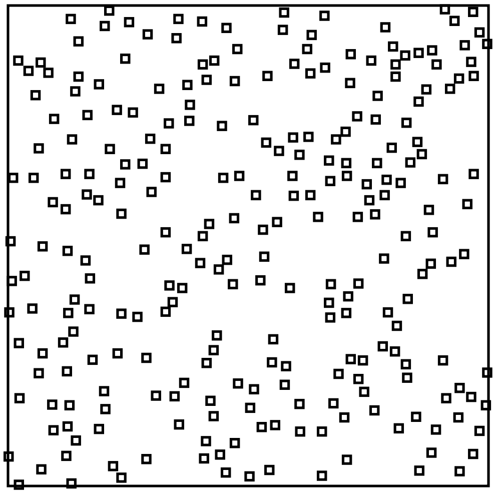

Figure 5.23: 256 Poisson disk sample points (minimum separation 0.025 pixel width)

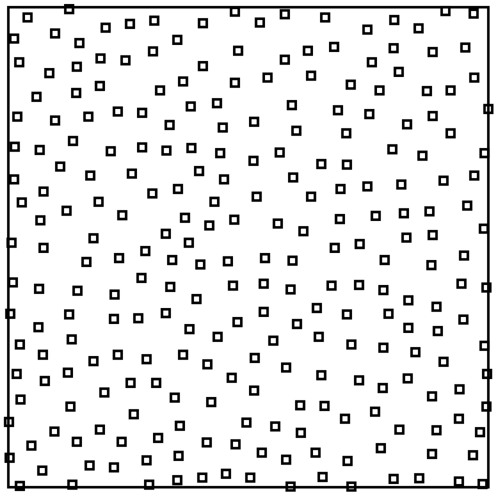

Figure 5.24: 256 Poisson disk sample points (minimum separation 0.05 pixel width)

As can be seen, there is less clumping for large minimum separation, but too large a separation can cause ghosting or even make sampling impossible.

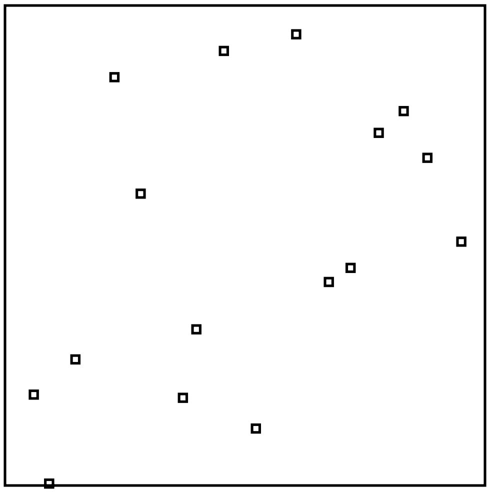

Figure 5.25: 16 separately (uncorrelated jitter) generated sample points

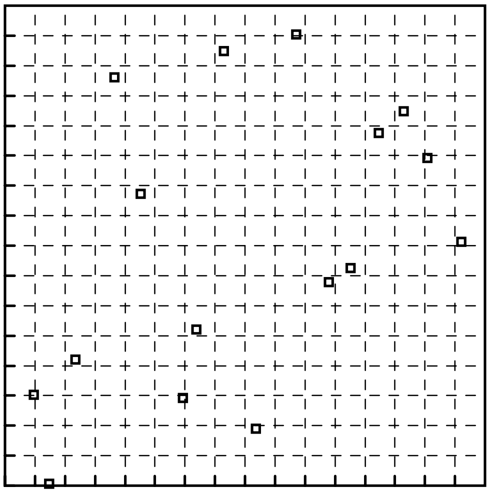

Figure 5.26: 16 separately generated sample points with guidelines

5.5.5 N-rooks Sampling

As discussed in Section 5.4.1, each of the dimensions of a sampling space can be separately partitioned and the strata of each dimension can be randomly associated so that each row and column of a grid will have one sample. A set of 16 samples of this type are shown in Figure 5.25. The underlying pattern of the samples is hard to see, unless a grid is superimposed as shown in Figure 5.26. A particularly descriptive name for this strategy is *N-rooks sampling*, because a acceptable set of sample cells will be the squares of an N by N chessboard with N rooks that

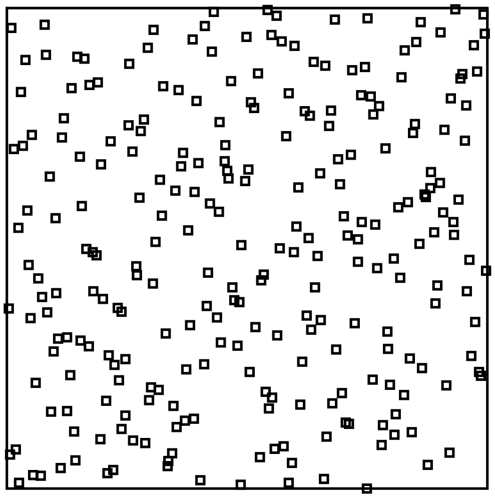

Figure 5.27: 256 separately (uncorrelated jitter) generated sample points

cannot capture each other in one move. There are two advantages of this type of sampling over conventional jitter. The first is that any number of samples can be taken with uncorrelated jitter, while conventional jitter usually requires a $m \times m$ or $m \times n$ pattern. The second is that if the sampled field varies only in one dimension, that dimension is fully stratified (just as in traditional distributed ray tracing).

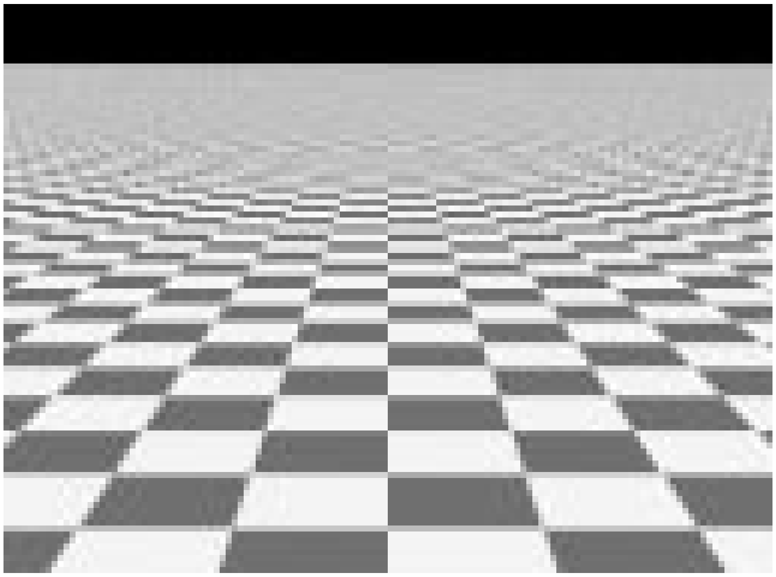

Figure 5.28: Test figure CHECKER

5.5.6 Empirical Behavior of Sampling Strategies

Each of the previous sampling methods was tested with 16 samples per pixel on each of four 128 by 96 pixel test images. All of the images were sampled with a one pixel width box filter. The first image, called CHECKER, shown in Figure 5.28, is that of an infinite checkerboard. Since the uncorrelated jitter is well suited to horizontal and vertical lines, the second test figure, CHECKER-45 (Figure 5.29), is the same checkboard rotated 45° to avoid such lines. The third figure, BALL (Figure 5.30), is a ball lit by two area light sources. The final figure, ALL (Figure 5.31), has a ball, a specular mirror, a glossy mirror, and a finite aperture lens. All four of the figures shown were sampled with 400 jittered samples, which is sufficient to produce an image with relatively small error.

Some of the sampling methods produced regular errors that were visually disturbing. On the checkerboards this was particularly true; the regular and semijittered sampling had visible aliasing artifacts. Figure 5.32 shows the regular sampling of CHECKER-45. The banding

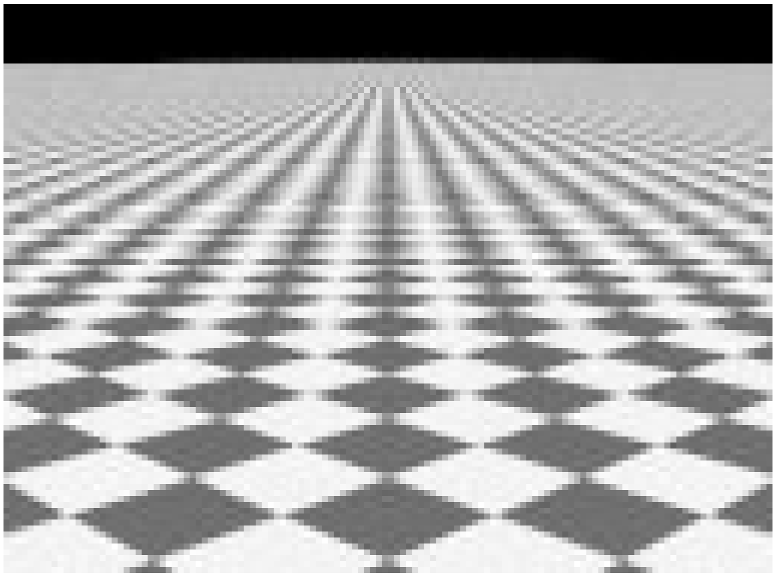

Figure 5.29: Test figure CHECKER-45

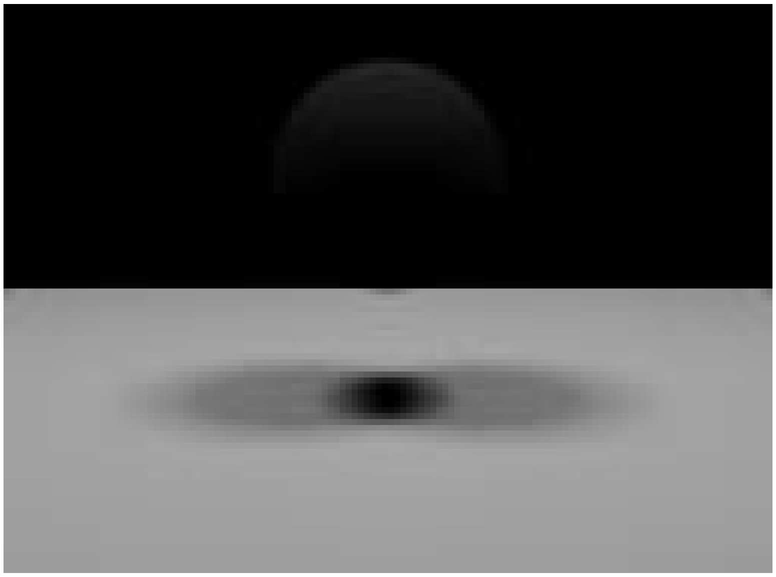

Figure 5.30: Test figure BALL

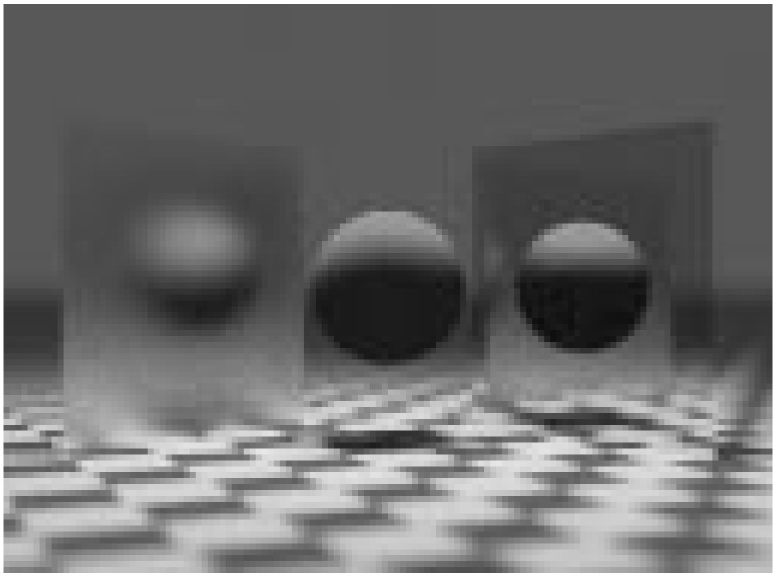

Figure 5.31: Test figure ALL

near the horizon is not present in the same image in the separately sampled image shown in Figure 5.33. This is an example of the common problem that it is the regularity of the error, rather than its magnitude, that is objectionable.

The average absolute error in luminance for each sampling strategy is listed for each test image in Tables 5.1 through 5.4. Surprisingly, the separate (uncorrelated jitter) sampling performed best by the average error metric on three of the four images, and on test image ALL was only outperformed by regular and half-jittered strategies, both of which are prone to aliasing. The standard deviation and maximum error caused by separate sampling also performs well relative to the other sampling strategies.

5.5.7 Theoretical Behavior of Sampling Strategies

In the last section it was demonstrated that for some images the separate sampling strategy performs quite well, even compared to Poisson disk sampling. Overall, the performance rankings of the strategies (from best to worst) was approximately separate, jittered, half-jittered, regular,

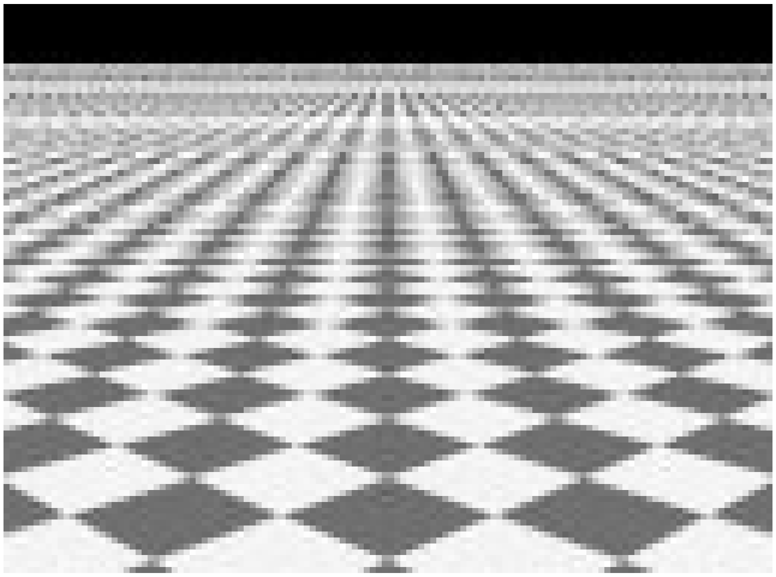

Figure 5.32: Test figure CHECKER-45 with 16 regular samples per pixel

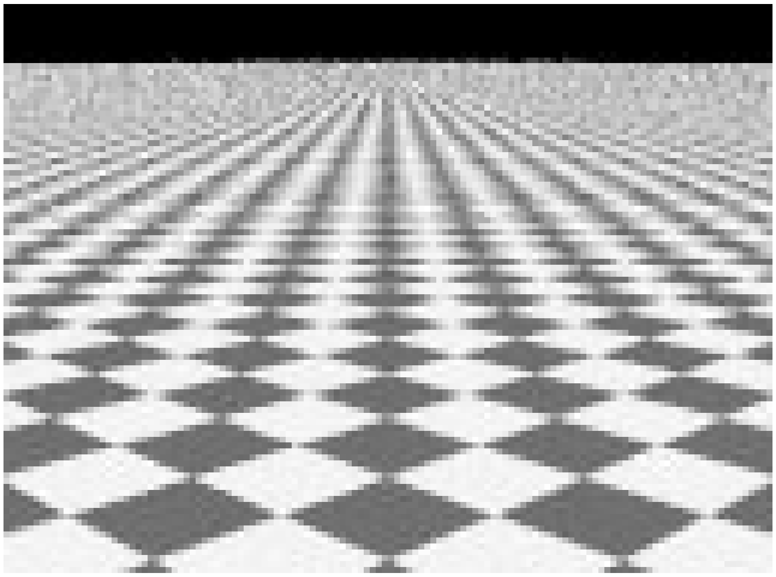

Figure 5.33: Test figure CHECKER-45 with 16 separate samples per pixel

sampling method	*ave*($\|E\|$)	*SD*($\|E\|$)	*max*($\|E\|$)
separate	0.0163	0.0303	0.308
jittered	0.0216	0.0368	0.394
poisson ($d = 0.2$)	0.0221	0.0362	0.334
poisson ($d = 0.1$)	0.0259	0.0413	0.308
half-jittered	0.0263	0.0437	0.368
random	0.0303	0.0479	0.331
regular	0.0312	0.0526	0.390

Table 5.1: Pixel errors in luminance for CHECKER

sampling method	*ave(|E|)*	*SD(|E|)*	*max(|E|)*
separate	0.0190	0.0296	0.286
jittered	0.0221	0.0340	0.291
poisson ($d = 0.2$)	0.0225	0.0343	0.333
poisson ($d = 0.1$)	0.0281	0.0416	0.304
half-jittered	0.0237	0.0445	0.410
regular	0.0273	0.0532	0.450
random	0.0315	0.0466	0.294

Table 5.2: Pixel errors in luminance for CHECKER-45

sampling method	*ave(|E|)*	*SD(|E|)*	*max(|E|)*
separate	0.00324	0.0099	0.179
regular	0.00363	0.0113	0.177
half-jittered	0.00365	0.0114	0.151
jittered	0.00370	0.0110	0.160
poisson ($d = 0.2$)	0.00404	0.0123	0.170
poisson ($d = 0.1$)	0.00526	0.0180	0.226
random	0.00607	0.0214	0.266

Table 5.3: Pixel errors in luminance for BALL

sampling method	*ave(|E|)*	*SD(|E|)*	*max(|E|)*
regular	0.0137	0.0235	0.242
half-jittered	0.0139	0.0234	0.205
separate	0.0148	0.0246	0.245
jittered	0.0150	0.0251	0.259
poisson ($d = 0.2$)	0.0156	0.0256	0.247
poisson ($d = 0.1$)	0.0172	0.0284	0.236
random	0.0190	0.0315	0.287

Table 5.4: Pixel errors in luminance for ALL

sampling method	*ave(D)*	*SD(D)*	*max(D)*
separate	0.162	0.0237	0.229
half-jittered	0.184	0.0187	0.243
jittered	0.193	0.0288	0.291
poisson ($d = 0.2$)	0.196	0.0332	0.290
regular	0.234	0.0000	0.234
poisson ($d = 0.1$)	0.245	0.0447	0.357
random	0.282	0.0557	0.428

Table 5.5: Discrepancies of different sampling strategies

sampling method	*ave(S)*	*SD(S)*	*max(S)*
half-jittered	0.0463	0.00290	0.0537
separate	0.0467	0.00847	0.0812
jittered	0.0495	0.00192	0.0678
poisson ($d = 0.2$)	0.0540	0.00891	0.0844
regular	0.0600	0.00000	0.0600
poisson ($d = 0.1$)	0.0743	0.02140	0.1740
random	0.0877	0.02390	0.2080

Table 5.6: Root mean square discrepancies of different sampling strategies

Poisson disk, and finally random. It is possible that these results are closely tied with filter choice, and that idea merits further investigation. The poor performance of Poisson disk goes against conventional wisdom. It would be nice to establish a quantitative metric for predicting the value of a particular strategy. It would also be a good idea to understand what we have done by using sample sets that are not strictly random, since presumably we are doing a Monte Carlo integration. In numerical integration theory these non-random sample sets are called *quasi-random*, because they have some statistical qualities that make them acceptable substitutes for true random samples. Zeremba developed a theory to bound the error of an integration based on equidistribution properties of the sample set (assuming certain continuity properties of the integrand)[126]. In this section, Zeremba's equidistribution metric, *discrepancy*, is discussed in the context of the sampling strategies from the last section.

sampling method	*ave(T)*	*SD(T)*	*max(T)*
separate	0.2132	0.0236	0.275
jittered	0.2555	0.0397	0.428
poisson ($d = 0.2$)	0.2613	0.0459	0.390
half-jittered	0.2608	0.0282	0.338
poisson ($d = 0.1$)	0.2921	0.0503	0.513
random	0.3434	0.0540	0.485
regular	0.3600	0.0000	0.360

Table 5.7: Stroud's discrepancies of different sampling strategies

The concept behind discrepancy is that we'd like a number that is small for very equidistributed sample sets, and large for poorly distributed sets. Imagine a set of N sample points, (x_i, y_i) on the unit square. Given a point (a, b) on the square, the set of points (x, y) such that $x < a$ and $y < b$ will define a smaller square (with lower left corner $(0, 0)$ and upper right corner (a, b)) with area ab. Let n be the number of the N sample points that falls within that smaller square. If the sample points are reasonably equidistributed, we would expect n/N to be about ab. Zeremba uses this observation to define the discrepancy, D, as the lowest upper bound of $|n/N - ab|$ for all (a, b). The average discrepancies of 100 sets of 16 samples for the various strategies are shown in Table 5.5. The table shows that the discrepancy of the separate samples is lowest, and the other sampling strategies are ranked in an order reasonably consistent with empirical behavior.

Zeremba points out that instead of taking the lowest upper bound of $|n/N - ab|$, we could take its root mean square value. The root mean square discrepancy, S, is shown in Table 5.6. Under this metric, the half-jittered sampling strategy slightly outperforms separate sampling. Again, the ordering is reasonably consistent with the observed behavior of the strategies.

Stroud has a slightly different definition of discrepancy: the discrepancy, T, is the lowest upper bound of $|n/N - (a-c)(b-d)|$, where c and d are the lower corner of the square, and n is the number of points within the square[109]. In other words, all squares are used, rather than

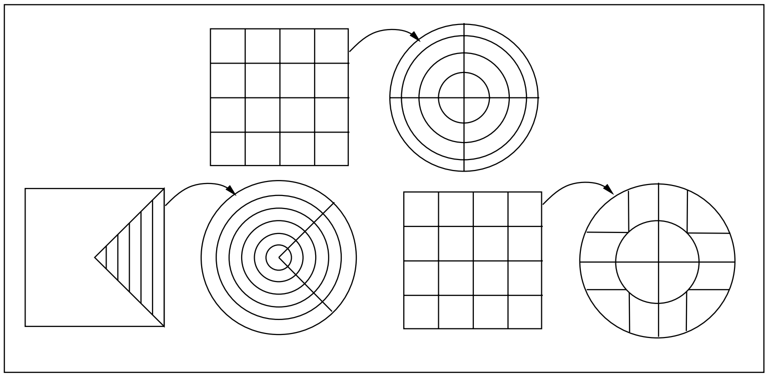

Figure 5.34: Two different ways to partition a circle

just squares with one corner at the origin. This makes the discrepancy of a 90° rotation of a set of points invariant. Stroud's discrepancy for 100 sets of samples is shown in Table 5.7. Applying this definition of discrepancy leads to an evaluation of sampling strategies that accords closely with the observable degree of error.

These tests indicate that discrepancy may be a useful tool in evaluating sampling strategies. One shortcoming of Zeremba's definitions is that it assumes a square domain. As shown in the top of Figure 5.34, a straight transformation from a evenly partitioned square to polar coordinates on a circle can stretch the strata. This implies that a good discrepancy in the canonical probability space does not guarantee good equidistribution in the actual domain (such as the lens area). A special purpose transformation is shown in the bottom of Figure 5.34. This keeps the strata from distorting too much. Unfortunately, the definition of discrepancy does not extend to non-square domains, so the only justification for preferring the bottom stratification of the circle is visual inspection and intuition.

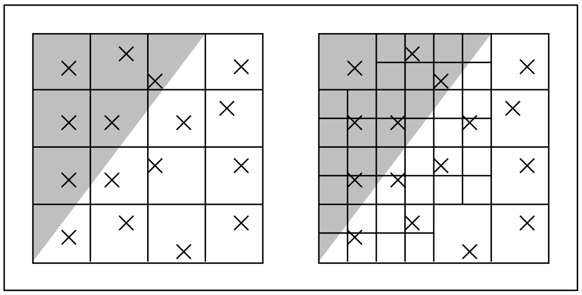

Figure 5.35: Adaptive subdivision applied when a sample has a different value than any of its 8-connected neighbors.

5.6 Strategies for Adaptive Sampling

One problem with taking a constant number of samples in each pixel is that different pixels can have vastly different variance. One technique is to take an initial sampling of the pixel with N samples, and apply more sampling if the initial samples vary[32, 30]. The tricky part of this technique is that if N is too small, a pixel with variance could have N samples that are the same (the classic feature detection problem[121]).

Kajiya tried to improve on adaptive sampling using a stratification of the samples[61]. This idea, which Kajiya called adaptive hierarchical sampling, is illustrated in Figure 5.35, where new samples are placed in strata adjacent to strata with different sample values. This is sort of a jittered Warnock subdivision. This basic idea has also been explored by Painter and Sloan[81].

Adaptive hierarchical sampling has not been successfully applied to distributed ray tracing. Kajiya[61] writes:

> *So far our experiments in finding adaptive criteria have not been terribly successful. We have not used adaption in computing the final images.*

I believe that the lack of success stems from the uncorrelated jitter used in distributed ray tracing; samples that are adjacent in the pixel probability space may not be adjacent in the

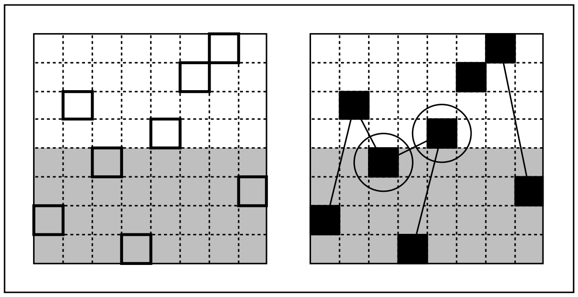

Figure 5.36: Adaptive subdivision applied to separately sampled space has many horizontal neighbors subdividing.

reflection or other probability space. As a simple example, consider the two dimensional separately sampled space shown on the left of Figure 5.36. If we want to sample hierarchically, we should refine in the two strata shown circled on the left of the figure. However, if we simply look at all the horizontal neighbor strata that are different, we will also subdivide in all the cells shown in black.

To investigate the feasibility of selectively subdividing only in the dimensions where there is actual variance, correlated jittering (where all probability spaces have the same connectivity) can be used with strict hierarchical subdivision. Figure 5.37 shows a test figure with 16 samples per pixel. Figures 5.38 and 5.39 show a detail of the figure before and after subdivision. The subdivision generated an average of about 5 extra rays per pixel over the whole image.

The subdivision technique was also applied to a path tracing application. Figure 5.40 shows indirect lighting coming off a specular block with 441 rays per pixel. Figure 5.41 shows the same picture with an initial set of 400 rays per pixel, and an average of 22 extra rays per pixel. The large number of initial samples is needed because at least one of the initial rays must hit the light source before extra sampling will take place.

These examples indicate that adaptive hierarchical sampling should be useful if the sampling spaces are subdivided in a reasonable way.

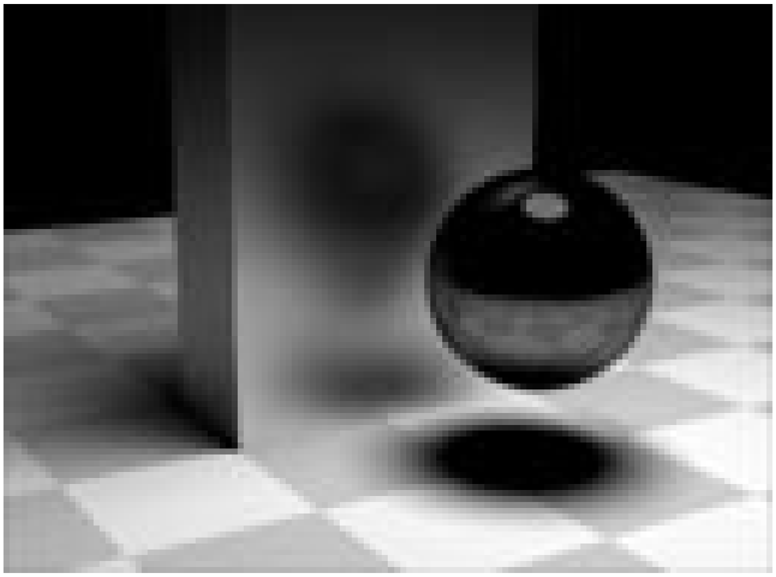

Figure 5.37: Test figure with glossy reflection

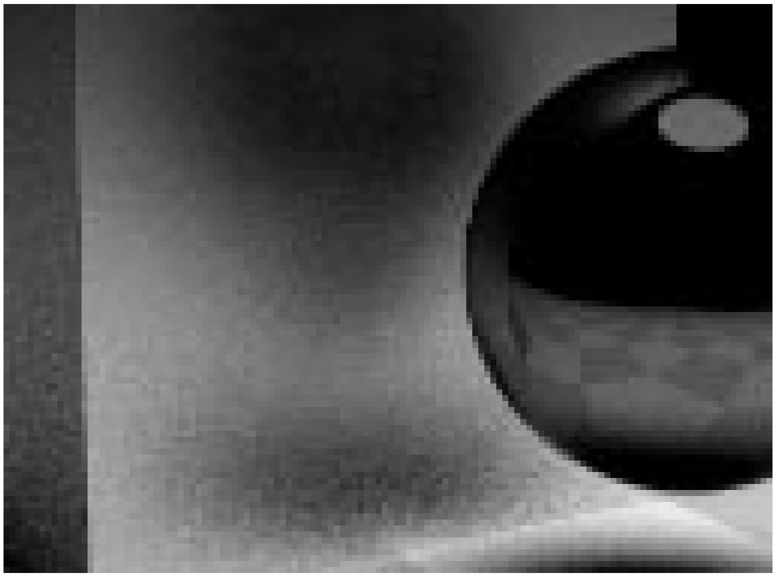

Figure 5.38: Detail of test figure with 16 samples per pixel

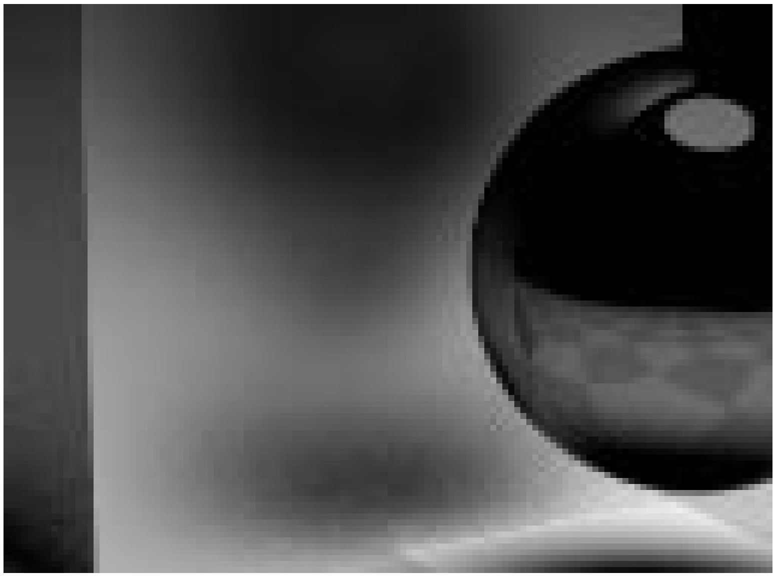

Figure 5.39: Detail of of test figure after two levels of subdivision

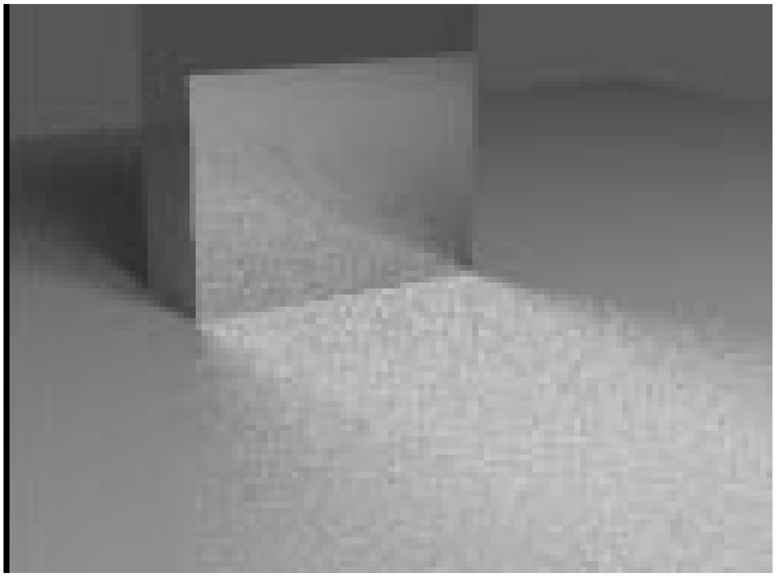

Figure 5.40: Test figure of light bouncing off mirrored block onto the ground

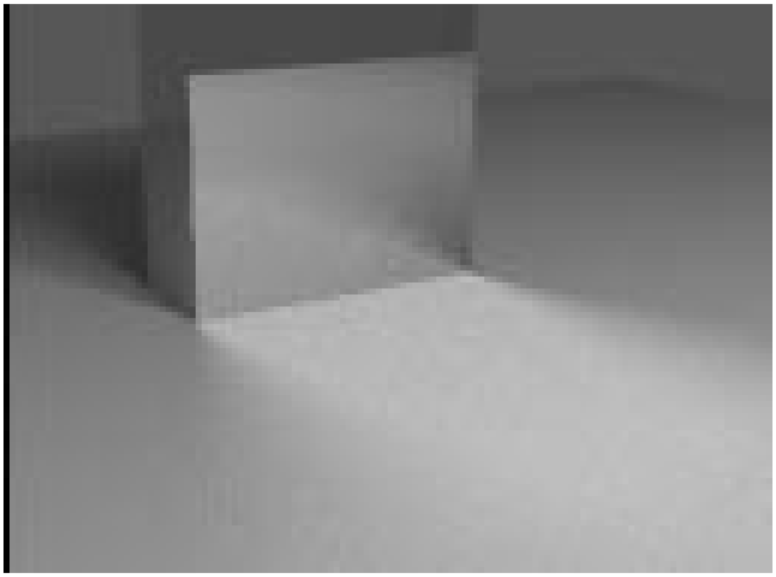

Figure 5.41: Test figure after one level of subdivision

5.7 Summary

Image-based methods calculate the colors for each pixel independently. A finite-aperture camera model fits nicely into this scheme. The color of each pixel is usually found by taking a weighted average of colors around the pixel center. The weighting function has certain features that restrict allowable functions. A new symmetric separable function was shown to satisfy these restrictions, and was also shown to have better filtering characteristics than the box filter.

Utah models assume only direct lighting plus an ambient term. Whitted-style ray tracing allows, in addition to direct lighting, ideal specular reflection and transmission, and gives a convenient way to test for shadowing.

Stochastic methods view the rendering problem as a Monte Carlo solution to a multidimensional integral. Cook et al.'s distributed ray tracing adds soft shadows, lens effects, and glossy reflection to Whitted Style ray tracing. The concept of uncorrelated jittering is central to distributed ray tracing, and can be understood as a general multidimensional sampling technique.

Kajiya generalized distributed ray tracing to allow for general light transport. Ward et al. modified Kajiya's method by saving information that can be used later on. A new stochastic method of restricting the number of shadow rays was discussed that makes creating images of scenes containing multiple light sources more practical.

If a predetermined number of samples is sent through each pixel, then the spatial distribution of samples should be chosen that minimizes error without introducing coherence to the distribution of error in the image. Separate sampling is shown to be superior to jittering or to Poisson Disk sampling for some cases. The notion of discrepancy, an error prediction metric used in numerical integration, is presented as a method of predicting the success of a sampling strategy. Using discrepancy seems to have some advantages over traditional signal processing approaches.

Kajiya's hierarchical sampling cannot be correctly used in stochastic rendering because of the uncorrelated jitter used to reduce noise. A modification to this sampling method was shown to allow an adaptive hierarchical sampling scheme for the full multidimensional integral.

CHAPTER 6

ZONAL SOLUTION METHODS

In zonal methods, the radiances of a scene are computed in advance of rendering in a view-independent process. In this chapter, zonal methods and their relation to image methods are discussed. In Section 6.1, the zonal method for diffuse environments is discussed, both in terms of linear systems of equations and in terms of physical simulation. Section 6.2 outlines optimization strategies for zonal environments, and speculates that the $O(N^2)$ time complexity of zonal methods can be beat. That section includes a proof that the expected number of rays needed for a zonal solution is $O(N)$. In Section 6.3, zonal methods for specular and glossy environments are discussed, and zonal and image-based methods are combined in a general way. This approach has the advantage over previous approaches to glossy environments that only the storage needed for each reflection type is required. Section 6.4 summarizes the content of this chapter.

6.1 Zonal Methods for Diffuse Environments

The simplest zonal methods assume all surfaces are diffuse reflectors[40]. First the environment is subdivided into N discrete patches that are assumed to be constant in reflectance, reflected power, and emitted power. The reflectance (R_i) and emitted power (Φ_i^e) are known, and the reflected power (Φ_i^r) is unknown. If we solve for Φ_i^r, then we can find Φ_i, the total power coming from the ith patch.

Once the total power of each patch is found, it can be converted to radiance using Equation 4.1. These radiance values can then be interpolated to form a smooth appearance[23]. The next several sections show methods of solving for Φ_i.

6.1.1 Diffuse Zonal Methods as Linear Algebraic Equations

The total power coming from the ith surface is the sum of emitted and reflected power: $\Phi_i = \Phi_i^e + \Phi_i^r$. The reflected power is the reflectivity times the incoming power. The incoming power is a fraction of the outgoing power of the other surfaces. The fraction of the outgoing power from surface *source* that hits surface *target* is called a form-factor (or view-factor or configuration factor), and is denoted $f_{source \to target}$. This yields an expression for the total power coming from surface i:

$$\Phi_i = \Phi_i^e + R_i \sum_{j=0}^{N} f_{j \to i} \; \Phi_j \tag{6.1}$$

Conservation of energy implies:

$$\sum_{i=0}^{N} f_{j \to i} \; \Phi_j \leq 1$$

with equality if the system is closed. Equation 6.1 can be written down in matrix form:

$$\mathbf{A}\Phi = \Phi^e \tag{6.2}$$

Where the matrix $\mathbf{A}$ is:

$$\mathbf{A} = \begin{bmatrix} (1 - R_1 f_{1\rightarrow 1}) & -R_1 f_{2\rightarrow 1} & -R_1 f_{3\rightarrow 1} & \cdots & -R_1 f_{N\rightarrow 1} \\ -R_2 f_{1\rightarrow 2} & (1 - R_2 f_{2\rightarrow 2}) & -R_2 f_{3\rightarrow 2} & \cdots & -R_2 f_{N\rightarrow 2} \\ -R_3 f_{1\rightarrow 3} & -R_3 f_{2\rightarrow 3} & (1 - R_3 f_{3\rightarrow 3}) & \cdots & -R_3 f_{N\rightarrow 3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -R_N f_{1\rightarrow N} & -R_N f_{2\rightarrow N} & -R_N f_{3\rightarrow N} & \cdots & (1 - R_N f_{N\rightarrow N}) \end{bmatrix} \tag{6.3}$$

It can be shown that the system rewritten in terms of radiance is diagonally dominant (though not sparse), so a Gauss-Seidel iterative method can be used to solve for Φ[23]. The Gauss-Seidel method will require $O(N^2)$ solution time, and the matrix $\mathbf{A}$ will require $O(N^2 f)$ initialization time, where f is the average time to calculate a form-factor. The storage requirement is $O(N^2)$ because $\mathbf{A}$ has N^2 elements.

6.1.2 Diffuse Zonal Methods as Light Transport Simulation

Another way to look at solving for Φ is to use direct simulation. We first set our estimate of Φ_i to be Φ_i^e for all i. For each surface i that has non-zero Φ_i^e, we can shoot a set of n_i energy packets each carrying a power of Φ_i^e/n_i. When a packet with power Φ hits a surface j, we can add $R_j\Phi$ for our estimate of Φ_j, and reflect a new energy packet with power $R_j\Phi$. This energy packet will bounce around the environment until it is depleted to a point where truncation is used. This basic energy packet tracing technique has been used in Heat Transfer[55, 28, 113], Illumination Engineering[108], and Physics[105, 58].

This method, which I call *reflection simulation*, has the problem that each reflection is followed by a ray intersection test to find the next surface hit. The later reflections will carry a relatively small amount of power, so tracing these later rays is somewhat wasteful in the sense that we have bad 'load-balancing': some rays do more work than others. One solution to this problem is to replace the reflection model with a model where light is absorbed and immediately reemitted (after attenuation by the reflectance). A scene where light is absorbed and reemitted in this way looks exactly like a scene where light is reflected, so solving for the transport in either model will yield the same solution.

To solve for the absorb and reemit model, we can again send power in bundles from light sources. When a bundle carrying power Φ hits a surface j, the absorbed power that will later be reemitted by surface j can be increased by $R_j\Phi$. After each light source emits its power, reflective surfaces can, in turn, emit their absorbed power. The efficiency of this method is best if surfaces with the greatest amount of power send their power first. This method, which I call *absorb and reemit simulation*, is used in computer graphics, where it is called *progressive refinement radiosity*[22]. The form factors needed to send energy from a given patch are usually calculated on the fly, so there is no $O(N^2)$ storage requirement. This space optimization could also be done in the Gauss-Seidel solution, since only one row of the matrix is used at a time.

6.1.3 Form Factor Calculation

If the absorb and reemit simulation method is used, the crucial step occurs when the designated source patch sends its power into the environment. The most straightforward method of sending this power is the Monte Carlo method[69, 1, 2, 100], where a random set of energy bundles is emitted (as rays) in a diffuse distribution, and these power carrying rays are sent to the other surfaces (generating rays in a diffuse distribution is discussed in Appendix B). This method is shown in Figure 6.1, where the grey source patch is sending many rays into the environment. Figure 6.2 shows a simple environment, similar to that used by [40, 72], with radiances calculated by the Monte Carlo method.

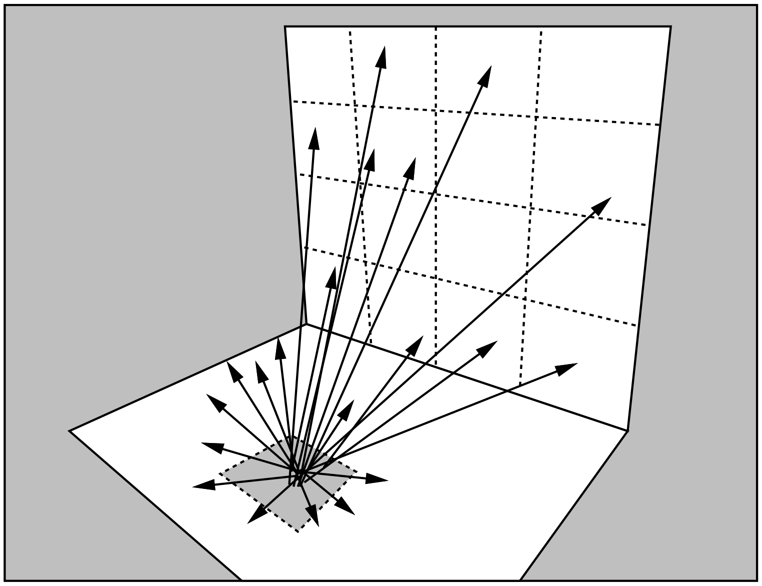

Figure 6.1: Monte Carlo emission of energy.

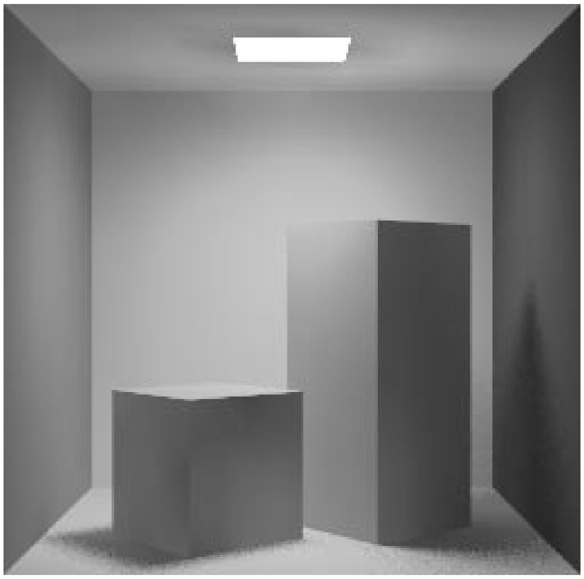

Figure 6.2: Zonal solution for diffuse scene.

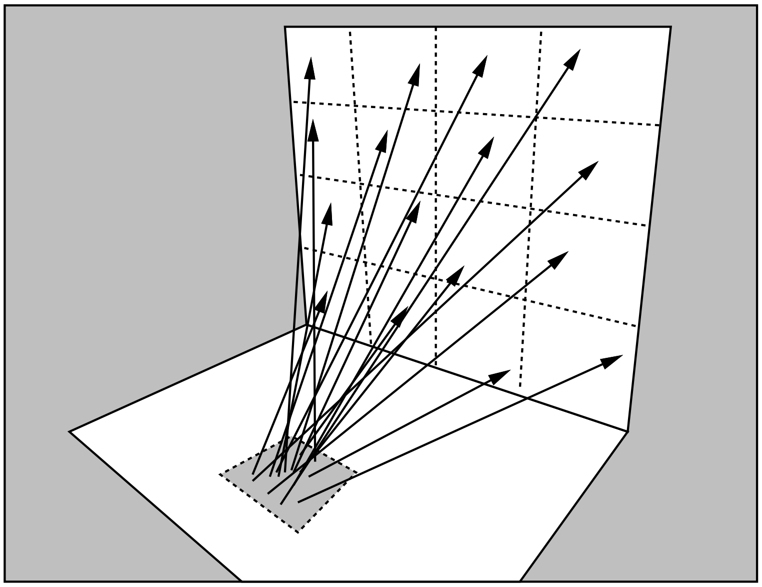

Figure 6.3: Analytic emission of energy.

Another way to send power is to explicitly calculate the energy sent from the source zone to every other zone, as shown in Figure 6.3. I call this way of transporting power an *analytic method.* If a ray between two patches is interrupted, then no power is sent between that pair. Wallace used this basic method combined with some optimizations and vertex oriented energy transport[118]. Another analytic method was used by Nishita and Nakamae who used shadow volumes to test for visibility[79]. A ray tracing-based analytic method has also been used in Illumination Engineering, though the method was restricted to rectangular zones aligned with the coordinate planes[15].

The classic way to send energy is by using a *Hemicube method*[23]. This method, shown in Figure 6.4, sends power into directional bins on the surface of a cube surrounding the zone. All of the energy sent through a directional zone goes to whatever patch is first seen through the center of the zone. Because of this the Hemicube is prone to aliasing. Artifacts arising from aliasing of the Hemicube can be lessened by increasing the Hemicube's directional resolu-

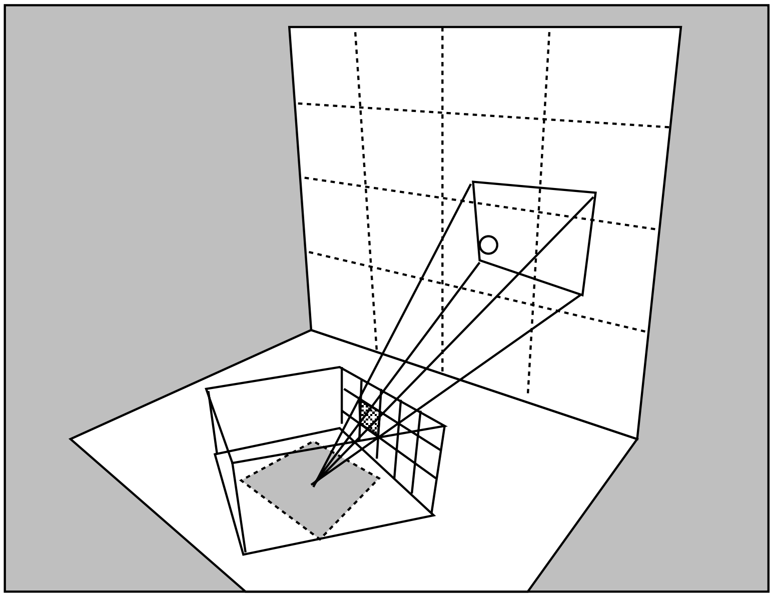

Figure 6.4: Hemicube emission of energy.

tion, rotating the hemicube by a random angle about the surface normal of the source patch, or employing correction techniques such as those presented by Baum et al.[8]. Methods of accelerating the Hemicube method by using hardware features, spatial coherence, and pixel coherence, are discussed by Rushmeier et al.[90]. One problem with the Hemicube method is that it approximates the parent patch as a point, so if a 'sending' scheme is used, shadows will be sharp.

The Hemicube and Monte Carlo methods of transporting power can be said to be in a family of methods that divide energy into angular bins. Other methods that do this are the ray tracing method of Sillion and Puech[103], and the Hemisphere method of Spencer[107]. Rather than sending power in directions, the analytic methods send power explicitly between each pair of zones. The advantage of the directional methods is that the amount of precision they employ is proportional to the solid angle subtended by the target patch. This avoids wasting much time

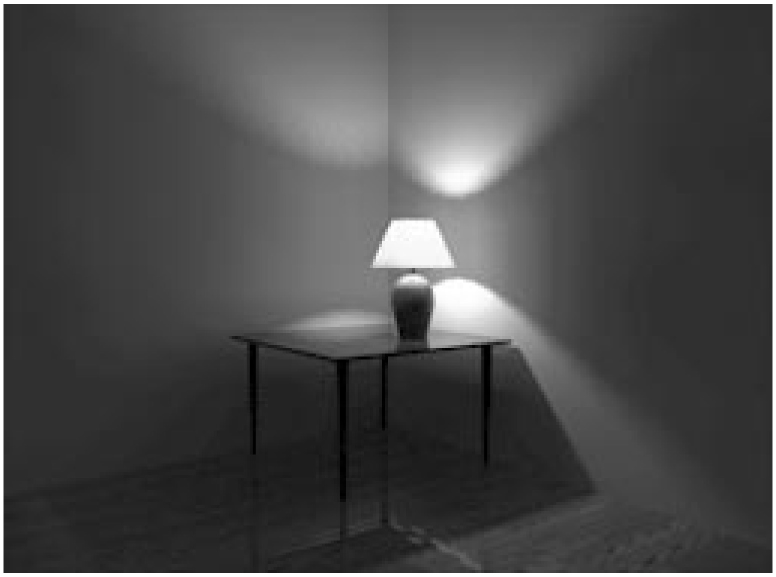

Figure 6.5: Zonal calculation with diffusely transmitting lampshade.

on small, far away patches. On the other hand, the error is not nearly as easy to predict as it is with analytic methods.

All of these methods could be used for diffuse transmission, as done by Rushmeier and Torrance[94]. An image with a diffusely transmitting lampshade is shown in Figure 6.5.

6.2 Optimizations for Diffuse Zonal Methods

Generating an image by the simulation methods of the last section require $O(Ns)$ where N is the number of emitting zones, and s is the amount of time it takes for one zone to send it accumulated power. This is because the solution will have an acceptable average error after a set number of reflections of light (usually 4 to 20 depending on average reflectivity in the scene), and each full set of reflections is approximated by all N zones firing their power once. Two basic optimizations are to reduce the number of patches N that send power, and to reduce the time s spent sending the power.

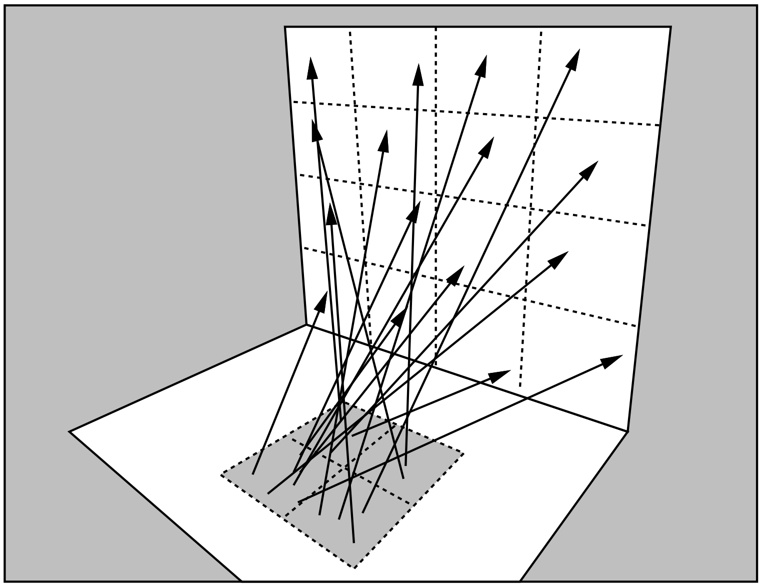

Figure 6.6: Four elements are collected into one patch before sending power.

6.2.1 Patch and Element Substructuring

The oldest optimization in zonal methods is *patch and element substructuring*[24]. Because indirect lighting is often soft, i.e. it does not change much in character over a distance, we can calculate some of this lighting with decreased accuracy. One way to do this is to collect several small zones, or *elements*, into one large *patch* which emits the accumulated power of the group of elements. The softness of the indirect component is shown in Figure 6.7, while the direct lighting can be hard, as shown in Figure 6.8. An emitting patch made up of four elements is shown in Figure 6.6. If zones can be constructed out of sets of e elements, and N is the total number of zones, then the time complexity can be reduced from the naive case of $O(Ns)$ to $O(Ns/e)$. Another speedup is to only use the elements for direct lighting, so the indirect lighting will have a reduced number of receiving zones[100]. Hanrahan and Salzmann have recently introduced a generalization of this idea where the patches are divided hierarchically into elements, and the level of size used to account for transport between two patches is based

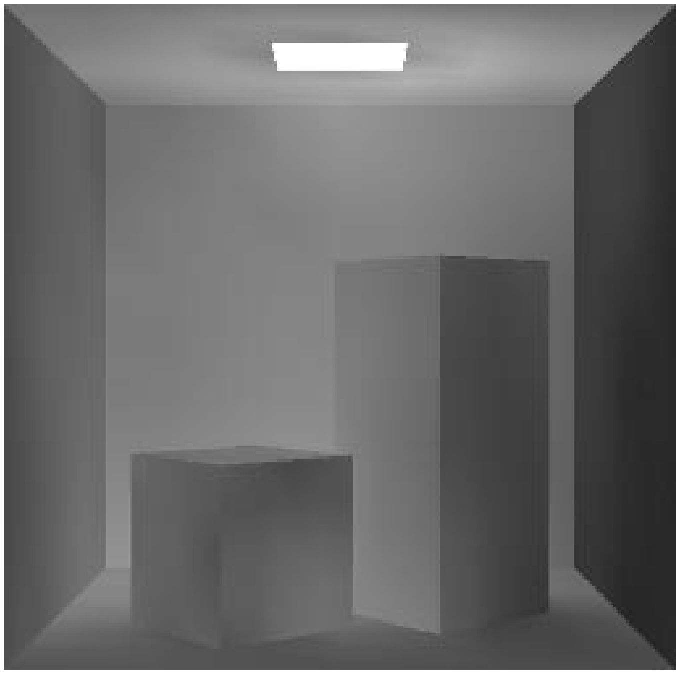

Figure 6.7: Indirect illumination.

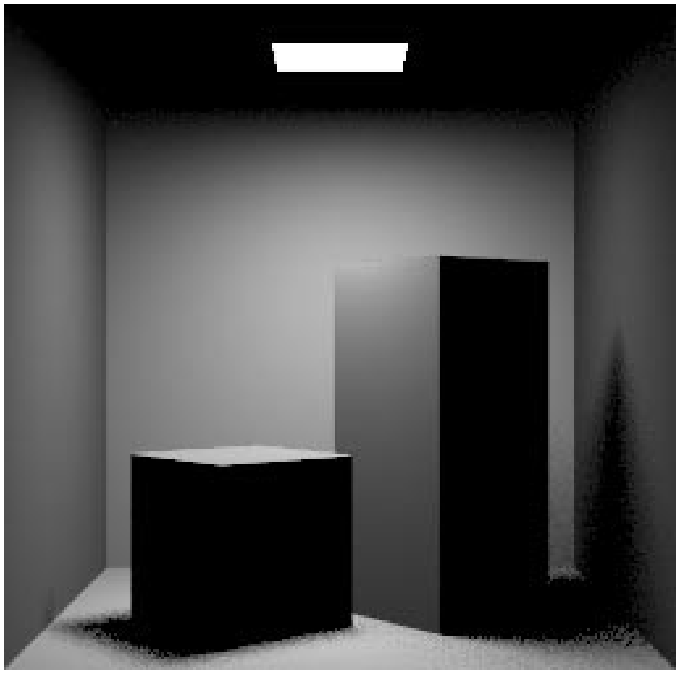

Figure 6.8: Direct illumination.

on the amount of power they exchange[48]. Though they have not yet extended their method to environments with occlusion, their initial results are very promising. Campbell and Fussell have extended adaptive meshing to non-quadtree data structures[17], and their initial results are accurate for shadows.

To get some insight into why the substructuring idea works, imagine that we are figuring the radiance at a point $\mathbf{x}$ that is due to a collection of elements. This is simply:

$$L(\mathbf{x}) = \int_{elements} \frac{R(\mathbf{x})}{\pi} L_{in}(\mathbf{x}, \psi) \cos\theta d\omega$$

We approximate this with a patch of radiance L_{ave} with the same solid angle as the elements. This amounts to approximating the cosine term with $\cos\theta_0$, where θ_0 is the angle to the center of the patch. If the patch is reasonable small, the maximum error will be small. In practice, if there are few regularities in the element radiances, the average error will be even lower.

A possible problem of all substructuring techniques is that if the initial discretization into patches is very fine, no subdivision may be needed ($e = 1$), so no speedup is attained.

6.2.2 Speeding Up the Emission of Power

The other way to speed up the zonal method is to reduce the amount of time it takes for a patch to emit power. The main way this is done is to reduce the accuracy of the solution if not much energy is being sent. Baum et al. did this by using a lower resolution Hemicube for indirect lighting[8]. Airey and Ouh-young used a Hemicube for direct lighting and then switched to ray tracing with the number of rays being proportional to the energy a patch has to send[1]. I used a strictly Monte Carlo method with the number of rays sent being set proportional to unsent power[100].

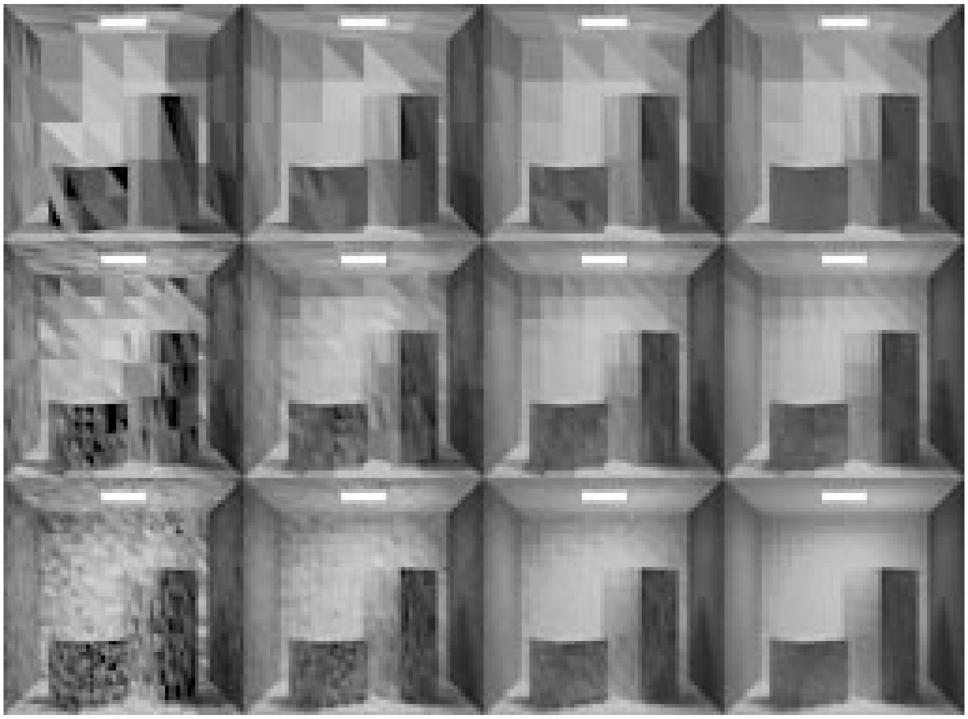

Figure 6.9: Rooms where each column has a number of rays proportional to the number of zones.

6.2.3 Optimal Time Complexity

Both Airey and I have empirically observed that the initial number of rays needed in the Monte Carlo approach is approximately proportional to the number of zones N. An example set of figures using this heuristic is shown in Figure 6.9, where the error does seem to go down consistently as the number of rays is kept proportional to the number of zones. This has the surprising implication that we can generate a zonal solution with $O(N)$ rays, so the solution time is approximately $O(\log N)$. This assumes that the average time to trace a ray is $O(N \log N)$, which is often true for divide and conquer search strategies in well-behaved scenes. However, the worst case behavior of ray tracing may be quite poor. Devillers[29] has done some initial work on the time needed to trace a ray, but it is still largely an unexplored topic. One unfortunate thing about sending rays in numbers proportional to power is that patch and element substructuring become useless; gathering four elements together will quadruple the total power, so no rays will be saved.

If Monte Carlo methods can achieve $O(N \log N)$ behavior, there is hope that a patch and element system might also. If the number of patches is $O(\log N)$, this will be true for the Hemicube method, and if the number of patches needed for a certain scene is constant (which I suspect may be true), then Wallace's ray tracing method could also be $O(N \log N)$, and the Hemicube method even better. One advantage of the Monte Carlo method is that the optimization seems to happen automatically, so no complicated substructuring or gathering of elements is needed.

6.2.4 Proof of Neccesary Ray Density

Suppose we are generating radiance values for a particular diffuse scene that has been discretized into zones. Assume that we are going to use Monte Carlo ray tracing to estimate the radiance values. In this section I prove that the expected number of rays needed for a solution is $O(N)$, where N is the number of zones, and these zones have certain properties. I will assume a very crude simulation, which is probably much less efficient than the emit and reemit strategy.

Rays will be independently emitted from light sources, each carrying the same amount of power (each of r rays will carry Φ/r power, where Φ is the total power). When a power carrying ray hits a zone, it is probabilistically absorbed or reflected. If reflected, its power is *not* attenuated. Attenuation is implicit because there is a $1 - R$ probability of extinction with each reflection. The ray continues to scatter throughout the environment until it is absorbed. Each ray is absorbed by exactly one zone. The amount of power reflected from the zone can be directly estimated as the power of the rays reflected by that zone. Or we could use the reflectance multiplied by the total power of absorbed and reflected rays. A third way would be to use the amount of power absorbed as an indirect estimate of the total power reflected. This is possible because the ratio of reflected to absorbed power is simply the ratio of reflectance to absorbance (one minus reflectance). In other words, the ratio of reflected to absorbed power is $R/(1-R)$. I will use this third scheme as our model because it simplifies the mathematics; any one power carrying ray is absorbed exactly once by exactly one surface, but might reflect from many surfaces, or reflect many times from a particular surface.

We would like to show that, given a desired variance bound for our radiance estimates, the expected number of rays traced in a simulation is $O(N)$, where N is the number of zones. If zero area zones are allowed, this will not be true because no rays will ever hit that zone. So we add the restriction that the ratio of the biggest to smallest zone area is bounded. To ensure termination of the physical process, we assume that reflectance is bounded by some number less than one. We also assume the environment we are zoning has some maximum radiance. This value will bound the average radiance of any zone, because the average of a set of values must lie within the range of the set.

Let the following definitions hold:

- N : number of zones.

- A : total area of all surfaces.

- A_i : area of ith zone.

- R_i : reflectance of ith zone.

- R_{max} : maximum reflectance in environment.

- Φ : total emitted power from all zones.

- Φ_i : outgoing reflected power for ith zone.

- L_i : outgoing reflected radiance from ith zone.

- L_{max} : maximum reflected radiance in environment.

- r : number of initial rays emitted by all zones.

We can relate some of these variables immediately. By definition, the radiance of a zone is:

$$L_i = \frac{\Phi_i}{\pi A_i}$$

We assume that the ratio of maximum to minimum zone area is bounded:

$$\frac{A_i}{A_j} < K$$

for all i and j, and for some constant K. This implies that for all i:

$$\frac{A}{KN} < A_i < \frac{KA}{N}$$

Further, we assume that there is some maximum radiance in the scene, L_{max}.

First, let's establish some useful relations. Suppose we have a sum, S, of N identically distributed random variables X_i, where each X_i is a value x with probabilty p and zero otherwise. We can immediately establish:

$$E(S) = E(\sum_{i=1}^{N} X_i) = NE(X_i) = Npx \tag{6.4}$$

and the variance of S is:

$$var(S) = var(\sum_{i=1}^{N} X_i) = Nvar(X_i) = N(E(X_i^2) - E(X_i)^2) = N(px^2 - p^2x^2) \leq Npx^2 \tag{6.5}$$

Initially we send r rays from the emitting zones. We want r large enough so that the variance in our estimate for every L_i, $var(L_i)$, is below some predefined threshold V_0. The reflected power from the ith zone is:

$$\Phi_i = \frac{R_i}{1 - R_i} \sum_{j=1}^{r} \phi_i^j$$

where ϕ_i^j is the amount of power absorbed from the jth ray by the ith zone. The sum is simply the total power absorbed by the ith zone. Since each ray is absorbed exactly once, the ith zone will either get all of its power (probability p), or none of it (probability $1 - p$). Because all rays are generated independently and according to the same distribution, p depends only on i, and not j, and will thus be denoted p_i. This means the expected value of Φ_i is (from (6.4)):

$$E(\Phi_i) = \frac{R_i}{1 - R_i} p_i \Phi$$

So the expected radiance of the ith zone is:

$$E(L_i) = \frac{1}{\pi A_i} \frac{R_i}{1 - R_i} p_i \Phi$$

Similarly, from (6.5) the variance of the power estimate satisfies the inequality:

$$var(\Phi_i) \leq \frac{R_i^2}{(1 - R_i)^2} p_i \frac{\Phi^2}{r}$$

and similarly:

$$var(L_i) \leq \frac{1}{\pi^2 A_i^2} \frac{R_i^2}{(1 - R_i)^2} p_i \frac{\Phi^2}{r}$$

We can also establish an upper bound for p_i. Since all radinaces are at most L_{max}.

$$E(L_i) = \frac{1}{\pi A_i} \frac{R_i}{1 - R_i} p_i \Phi \leq L_{max}$$

and thus:

$$p_i \leq \pi A_i \frac{1 - R_i}{R_i \Phi} L_{max}$$

So our bound for variance of L_i becomes:

$$var(L_i) \leq \frac{1}{\pi A_i} \frac{R_i}{1 - R_i} L_{max} \frac{\Phi^2}{r}$$

From previous bounds on A_i (all areas at least $A/(KN)$), and R_i we have for all i:

$$var(L_i) \leq \frac{N}{\pi KA} \frac{R_{max}}{1 - R_{max}} L_{max} \frac{\Phi^2}{r}$$

Grouping the constants into a new constant C yields:

$$var(L_i) \leq C\frac{N}{r}$$

To ensure that we estimate all L_i with a variance lower than V_0, we can simply use:

$$r = \frac{CN}{V_0} \tag{6.6}$$

Since each original ray from the light source may reflect off many surfaces, the expected number of *total* rays, the number of intersection calculations needed, as opposed to the number of original power carrying paths, is r times one plus the average number of reflections for each packet. Because reflectivity is bounded, we have:

$$E(r_{total}) \leq r(1 + R_{max} + R_{max}^2 + R_{max}^3 + \cdots) = \frac{r}{1 - R_{max}}$$

So, choosing r according to (6.6) ensures that we will satisfy our target variance condition and:

$$E(r_{total}) \leq \frac{CN}{V_0(1 - R_{max})} = O(N)$$

Thus, the number of rays needed for this Monte Carlo simulation is $O(N)$, where N is the number of zones.

6.2.5 Gauss-Seidel Revisited

One optimization used by Cohen et al. in their progressive refinement method was to keep an ambient term that would make their average error approximately zero (though the absolute error would still be fairly large)[22]. Their steel mill picture was produced by doing a partial progressive refinement solution with an ambient term, followed by performing the following on each visible surface:

$$\Phi_i = \Phi_i^e + R_i \sum_{j=1}^{N} f_{j \to i} \ \Phi_j$$

This is just one full iteration (for the visible surface) of the Gauss-Seidel method, with a good initial guess. The ambient term is crucial for this to work, or the estimate would be too low. This technique might also work quite well if ray tracing were used, because the number of rays needed for one patch might be much smaller than the number of zones.

6.2.6 Division into Subsystems

Many scenes have subdomains that are disconnected, such as two separate rooms, or subdomains that are partially connected, such as two adjacent rooms with an open door. Some preliminary work on solving these subdomains separately and then linking the solutions has been done by Neumann and Neumann[78] and by Xu et al.[123]. These methods are promising because they will give a speedup for any solution method with time complexity higher than $O(N)$.

6.2.7 Calculation of Direct Lighting in Viewing Phase

Because indirect lighting is soft, a zonal calculation of indirect lighting can use fairly coarse discretization, and a view-dependent solution of the direct lighting can be performed with high accuracy. Both Ward[119] and I[100] have used this idea to produce images. One benefit of doing the direct lighting in the viewing phase is that bump mapping[10] can be used as shown in Figure 6.10. Recently, Chen and Wu have added bump mapping directly to the radiosity phase[18], but their method may behave poorly if sharp shadows fall across a bump mapped surface.

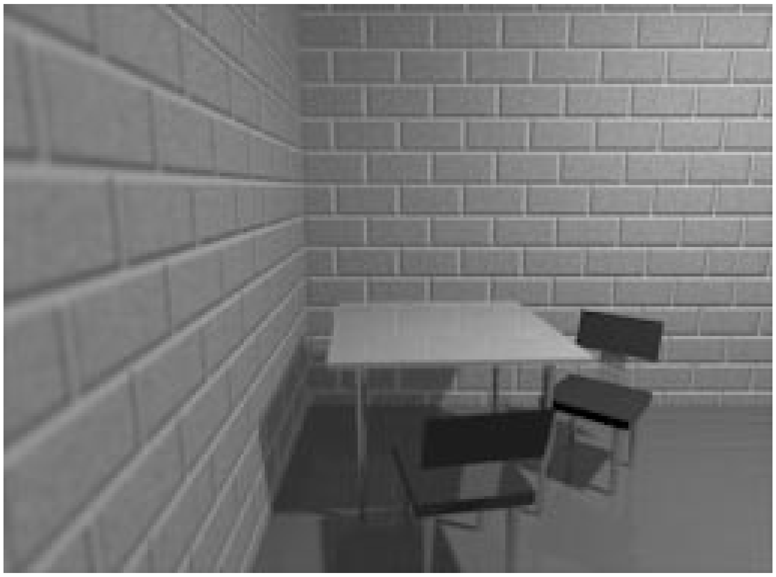

Figure 6.10: Radiosity with image based direct lighting calculation allows proper shading of bump maps.

6.2.8 One Level Path Tracing

Rushmeier developed a method where Kajiya's path tracing is used for primary rays, while secondary (reflected) rays get their radiance values by querying values from a coarse zonal database[92]. Again, this works because the indirect lighting is soft. This technique is appealing because all of the detail of the direct lighting is accounted for, and path tracing is used only while stratification can be maintained. This method brings up the interesting possibility of using an all diffuse zonal phase, while allowing glossy reflection in the viewing phase.

6.3 Zonal Methods for General Reflection Properties

Some of the zonal techniques discussed earlier can be extended to non-diffuse reflection types. The most important application is to scenes that include specular surfaces, but glossy surfaces are sometimes desirable too. In this section these extensions are discussed, and some generalizations of previous methods are developed.

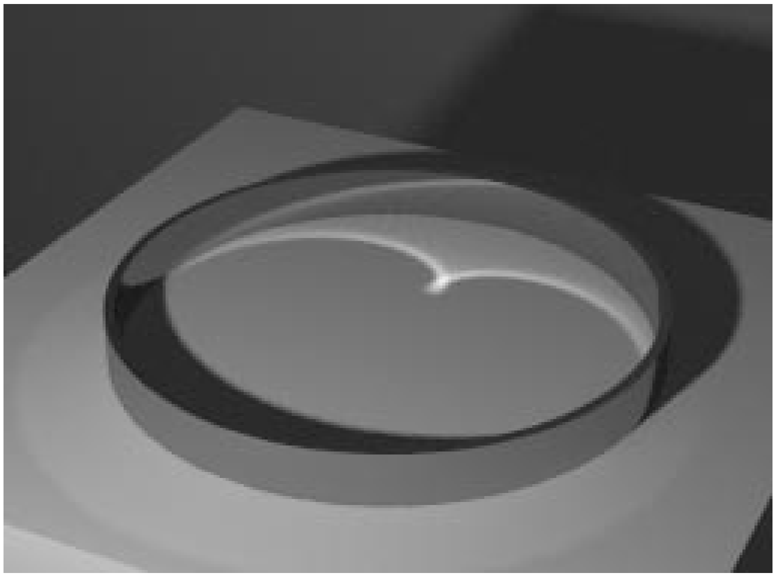

Figure 6.11: Zonal calculations allow for specular transport.

6.3.1 Including Specular Reflection

The simplest method of including specular reflection in a zonal calculation is the *image method*[117, 94]. In the image method, a specular surface is replaced by a hole into a virtual environment. This method works only for planer mirrors, but performs very well for environments that have one important specular surface like a mirror or highly polished floor.

Wallace used distributed ray tracing with a zonal preprocess that allowed highlights and reflections in a radiosity scene, but did not account for most specular transport in the zonal phase[117]. The visual quality of the images produced by this simple method implies that many scenes may not require specular transport in the viewing phase.

Arvo traced rays from the light source in a preprocess that simulated the light transport off specular objects that can cause caustics such as that shown in Figure 6.11[5]. As can be seen in the figure, indirect lighting involving specular surfaces is not always soft. More on this subject can be found in [100]. Heckbert[53] and Watt[120] have also extended Arvo's method.

Malley extended his Monte Carlo power transport method to account for zonal transport by specular surfaces[69]. He did this by allowing power carrying rays to reflect off specular surfaces as shown in Figure 6.12. The colors of specular surfaces can be determined in the viewing phase by standard ray tracing. Sillion and Puech used a similar technique to account for specular reflection, and included subdivision strategies for sampling more heavily where ray paths diverged[103]. An application of the Monte Carlo technique with specular transport is shown in Figures 6.13 and 6.14.

6.3.2 Including General Reflection Functions

The ideas used for specular zonal methods extend nicely to general reflection functions. We can take any set of surfaces, and model them as *reflectors*, rather than absorbers and reemitters as discussed earlier. In the Figure 6.12, the mirror is just acting as a reflector, and the diffuse reflectors are absorbing and then reemitting. There is no reason we cannot make any surface type either a reflector or reemitter, at our discretion.

Any reflectors will not store zonal values and must be assigned colors in the viewing phase. In Figure 6.15, the block on the right is a reflector, and its final colors are determined with path tracing.

Any non-diffuse reflectors can have zonal values, as long as each incoming power packet adds to a power *distribution function* that will be reemitted. In the viewing stage, this distribution can be queried with results depending on viewer position. The distribution functions could be stored in a Hemicube as done by Immel et al.[56], as spherical harmonics as done by Cabral et al.[16], or in hemispherical tables as done by Hall and Rushmeier[42] and Le Saec and Schlick[95].

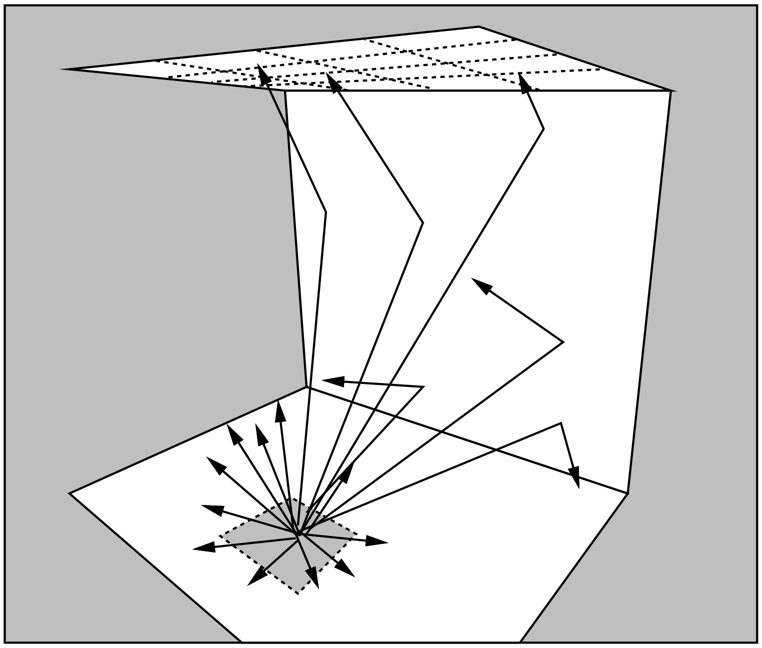

Figure 6.12: Monte Carlo emission of energy with specular reflection.

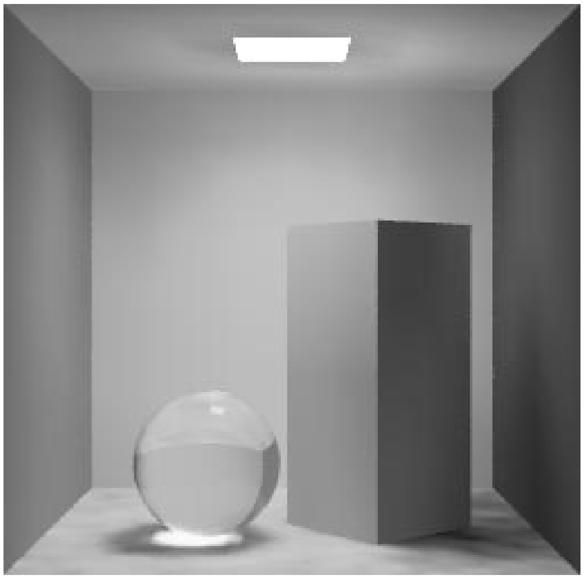

Figure 6.13: Zonal calculations allow for specular transport.

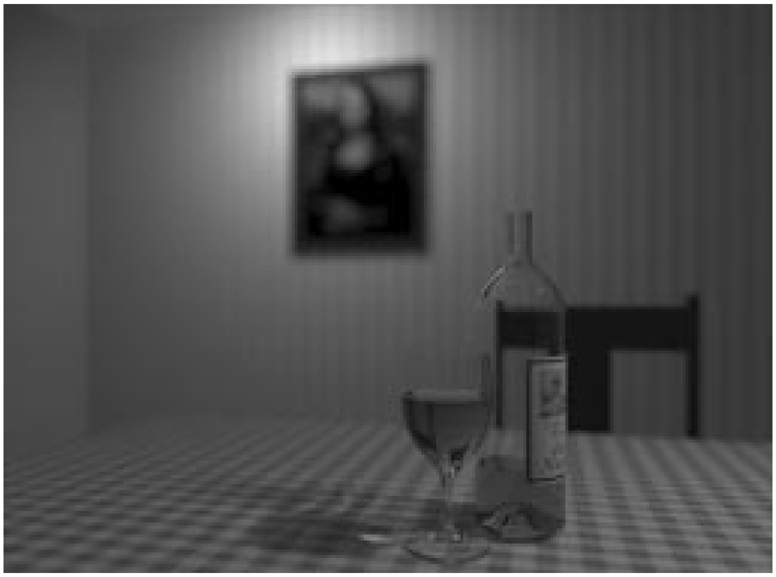

Figure 6.14: Zonal calculations allow for lighting through glass.

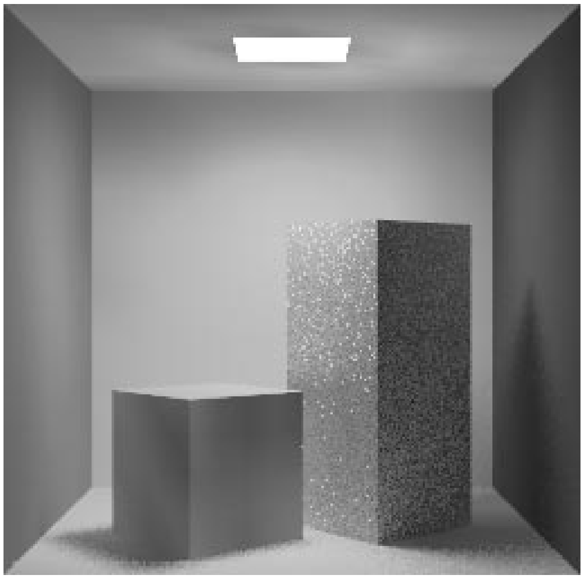

Figure 6.15: Block on right is reflector (not zonal). The rest of the scene is made up of reemitters.

The Monte Carlo method could be used by generating outgoing power rays according to the shape of the unemitted power function.

6.4 Summary

Zonal methods partition the surfaces in a scene into zones which are assumed to be of approximately equal radiance. In the case where all surfaces are diffuse, the radiances of all the zones may be written as a set of simultaneous linear equations and solved by conventional means. It is also possible to bypass forming this system of equations by doing an explicit physical simulation of light transport. This simulation approach can be straightforwardly generalized to scenes that include non-diffuse surfaces. A method was introduced that allows some surfaces to be broken into zones, while others are not. Some time complexity issues were discussed that make ray tracing based energy transport appear potentially better than other zonal strategies.

CHAPTER 7

EXTENSIONS TO BASIC SOLUTIONS

The previous chapters have assumed that the scene has vacuum between the surfaces, that the solution is needed at only one wavelength, and that the scene is static. This chapter addresses what to do when these assumptions do not hold.

Section 7.1 discusses zonal techniques for participating media, with emphasis on extending the Monte Carlo techniques of the last chapter. In Section 7.2, solutions for multiple wavelengths are discussed. Section 7.3 examines solutions for dynamic environments. Finally, the contents of the chapter are summarized in Section 7.4.

7.1 Including Participating Media

Solutions with primary scattering from participating media have been discussed by Blinn[11] and by Kajiya and Von Herzon[62]. There have also been many methods that use primary and secondary scattering in the Heat Transfer literature, but most of these make many simplifying assumptions about either geometry or scattering function[54]. A zonal method has been applied to the case of an isotropically scattering medium by Rushmeier[93]. Her approach was not based on the energy shooting strategy, so arbitrary scattering functions were not allowed. Rushmeier also developed Monte Carlo methods that fully account for arbitrary scattering functions, but these methods are not as efficient as the zonal method she used for the isotropic case[92].

The power shooting techniques of the last chapter can be extended to media with general scattering functions. As with surfaces, the media can be divided into zones, where each zone located at x_i is considered to contain small particles with number density N (number of particles per unit volume), scattering cross sectional area A, and a *SPF* $s(x_i, \psi_{in}, \psi_{out}, \lambda)$. Values of A and s for many real world gasses as well as an excellent treatment of scattering in general can be found in the paper by Klassen[67].

When a ray carrying power Φ passes a distance t through a zone, it will have some amount of power Φ_s scattered by the zone:

$$\Phi_s = \Phi(1 - e^{-NAt})$$

This energy can be entered into the $I(x_i, \psi, \lambda)$ table for the zone. During the shooting phase of the algorithm, volume zones can fire their accumulated power according to $I(x_i, \psi, \lambda)$.

After the zonal phase is finished, the radiant intensity functions can be interpolated to zone vertices. When a viewing ray passes through the media, it will pick up some of the radiance of the zones, and some of the background radiance will be attenuated.

If the radiant intensity functions $I(x_i, \psi, \lambda)$ of the zones are converted to $L(x_i, \psi, \lambda)$ radiance functions for the particles, then calculating the color seen along a viewing ray is determined by:

$$L(x, \psi, \lambda) = L_b(\psi, \lambda)e^{-NAt} + (1 - e^{-NAt})L(x_i, \psi, \lambda)$$

where the ray passes through a distance t of the zone, and L_b is the radiance seen through the zone. If N, A, or $L(x_i, \psi, \lambda)$ vary along the ray (which will happen if we interpolate between zone vertices), then ray integration can be done in small steps as done by Kajiya and Von

Herzen[62]. This machinery is also used for scalar volume visualization, though the lighting models used are not necessarily physically motivated[68, 114].

Because the volume zones are treated similarly to surface zones, it is straightforward to allow volume zones to be light sources such as fire.

One potential drawback that arises when volumes are used is that a volume and surface object may overlap. This means care must be taken when emitting energy from a volume element to avoid sending energy from the inside of an opaque object. A possible solution is to allow volume geometries that are not rectilinear, so the volumes can be wrapped around a surface. This unfortunately makes ray tracking more difficult[101]. Rushmeier has suggested that any power carrying rays be dissallowed if their origins are inside an object[92]. Her approach would at least prevent physically impossible rays.

As can be done with surfaces, we can calculate the indirect lighting of volumes with a low resolution zonal phase, and calculate the detailed direct lighting of the medium in the viewing phase. This is similar to calculating lighting on textured surface zones where average reflectance is used in the zonal calculation. With volumes, average number density and cross sectional area can be used.

7.2 Wavelength Dependencies

Because of the many perceptual issues involved, wavelength sampling and conversion to the proper format for display are perhaps the least well-understood part of the image generation process. Color in most graphics applications simply means three separate solutions for the red, green, and blue channels. What these channels really mean, and how they relate to spectral sampling is subtle[36]. Many authors have asserted that some form of spectral sampling, with later conversion to RGB is preferable[44, 71], especially for certain pathological cases. Just what degree of sampling is needed is unclear, but most practitioners use a small number of samples (4-20), and this seems to be adequate for many applications. Meyer has suggested a group of four wavelengths chosen with perceptual motivation[71], and these wavelengths have been used for the figures in this work.

In some situations, it is not possible to generalize to multiple wavelengths simply by maintaining an array of radiances for a selection of wavelengths. One obvious example is when refraction causes dispersion, so that the different wavelengths require different rays to represent their paths. However, *usually* we will want to treat the different wavelengths as a group, so that the number of rays traced is minimized. Thomas has simulated refraction and dispersion effects in a way that maintains coherence of the wavelengths until dispersion occurs, at which point an adaptive splitting occurs[112]. Thomas' approach is complicated to implement, but there is probably no easy method to capture dispersion effects while maintaining coherence.

Another situation where spectral coherence problems arise is when the general energy transport methods of Section 6.3 are used. If a zonal surface is glossy, the light leaving it may have very different directional distributions for different wavelengths. Because of this difference, there is no way to send out a bundle of rays while having the rays carry constant spectral quantities. There are two solutions to this problem. The naive, and inefficient, approach would be to send separate rays for each wavelength. A better approach would be to send rays based on the total power integrated over all wavelengths (or perhaps luminance), and to weight each ray according to the spectral distribution for its direction.

Some of the color effects caused by physical optics have been treated as special cases. Opalescence has been simulated using a simplified model of scattering[125]. Interference effects (e.g. thin-film effects) have been simulated using analytical models on surfaces where a priori knowledge that interference will occur exists[104]. Because interference produces almost pure spectral colors, care must be taken when displaying images containing interference effects, because they may be outside the gamut of the display device. Smits and Meyer have observed that the quality of the conversion depends on in what color space the extrapolation to a displayable color takes place[104]. Diffraction effects, such as the colored bands around headlights, have been modeled by Nakamae et al.[77].

The issue of wavelength sampling will only increase in importance. Currently, most input data to graphics programs is non-physical data that is often reasonably smooth. Once more varying spectral data, such as fluorescent light spectra, are used, the relation of spectral sample point location to the peaks and valleys in the spectral function will become more important. It is unclear what the resolution to this issue will be, but it is possible that for many applications the efficient solution of smoothing the input spectral data to minimize the number of spectral samples will be acceptable.

7.3 Time Dependencies

When the image is formed over a time interval, and objects are moving in that interval, blurring and dynamic shading can occur. This can be handled by image based methods in a fairly straightforward (but not that easy to implement) manner. This is done by integrating over time in addition to pixel area, lens area, reflection direction, and light source areas. Distributed ray tracing accomplishes this by associating a time with each ray, thus sampling time as another dimension of the integrand[26, 25].

Zonal solution methods do not adapt so easily to time dependent solutions. One of their fundamental assumptions is that the system is either solved for in a very small time period, or is in steady state. Several researchers have looked into solving for a given configuration and then modifying this solution after changes in the geometry of emitted energy distributions take place[9, 88, 19]

An approximate solution method for dynamic environments would be to solve for set time intervals and linearly interpolate for times between these intervals. A viewing ray at a certain time could approximate a zonal value for a given time. This method would be more robust if the direct light calculations were image based.

7.4 Summary

The ray tracing based energy transport techniques of Chapter 6 can be generalized to include volumes of media.

When wavelength dependencies are added into image based methods, the wavelength information can be carried along with each ray. This idea breaks down when dispersion of different wavelengths occurs at a surface. In this case the ray can be split into rays representing different spectral bands. Exactly how one should handle spectral sampling is an open issue, but some small number (4-20) of samples usually is sufficient.

When wavelength dependencies are added to zonal methods, the spectral information can be stored as a group for each patch. A complication arises in the case where the **BRDF** of a zone depends on incident directions. In this case, the luminance of the radiant intensity distribution can be used as a distribution for energy carrying rays sent by the zone.

When the geometry or lighting in a scene are time dependent, stochastic image based methods can treat time as just another dimension to be integrated over. Time dependency in zonal methods is much more of a problem. If the time dependency can be localized, then the partial zoning method of Chapter 6 could be used to find an approximate solution. Another possibility is that separate zonal solutions can be made at specified sample points, and approximate solutions could be found using interpolation.

CHAPTER 8

IMPLEMENTATION

All of the raster pictures in the previous chapters were produced using a moderate size C++ program (approximately 9000 lines). The program follows three basic phases: the input of the geometric and viewing parameters, the zonal transport of power, and the formation of an image for a specified set of viewing parameters.

Two basic techniques associated with object oriented programming were used in designing the code. The first was the implementation of a set of 'utility' classes, such as points and vectors. The second was the use of subclassing (inheritance) to generate cleaner, more extensible code than usually results from conventional programming practice.

Section 8.1 outlines the utility classes implemented, and how these classes were implemented. The ray tracing primitives are described in Section 8.2. The material property classes are discussed in Section 8.3. The Zonal calculation phase of the code is described in Section 8.4. The viewing phase is then covered in Section 8.5. Finally, this chapter is summarized in Section 8.6.

8.1 Toolbox Classes

Some utility classes were obviously needed: points, vectors, 4 by 4 transformation matrices, and colors. Operations involving these classes are defined using operator overloading mechanisms. Only operations that have mathematical meaning are allowed as suggested by Ronald Goldman. The allowed expressions are:

1. vector + vector = vector
2. vector - vector = vector
3. point + vector = point
4. vector + point = point
5. scaler * vector = vector
6. vector * scaler = vector
7. point - vector = point
8. rotation-matrix * rotation-matrix = rotation-matrix
9. point * rotation-matrix = point
10. vector * rotation-matrix = vector

Denying operations such as the addition of two points makes mistakes in expressions easier to catch. Unfortunately, it also makes it difficult to take the centroid of a set of points. In our experience, the benefits of these operational restrictions have greatly outweighed the deficits.

It is often useful to restrict a vector to be of unit length, such as when dealing with direction vectors in a ray. The unit-vector class has been defined as a subclass of vector. The unit-vector class can be confusing because some operations such as a matrix multiplication can change its length. A partial resolution to the problems of rigor versus efficiency is to let the unit-vector inherit all of the operations of the vector type. For example, we can have 'vector = vector + unit-vector'. A unit vector can be initialized by an assignment operator 'unit-vector = vector', where the assignment occurs after automatic normalization of the vector value. Unit vectors themselves have a subclass, surface-normal-vector, which responds to transformations differently than unit-vectors.

Colors have the operators of addition, multiplication, division, and subtraction with themselves, and multiplication and division with scalars. Initially, a RGB color model was used. This was later replaced by a very general spectral model, where each color was approximated by a piecewise linear approximation with arbitrary node locations. When two colors were combined by an operation, a new color with possibly more nodes was generated. This meant that unbounded lists had to be used to store the node locations and amplitudes. Though this general color representation was well suited for complicated spectral distributions, such as the emission spectrum of a fluorescent light, and was spatially efficient for simple spectra, the time needed to manage the color variables was too large to justify the switch from RGB (typically, the run time more than doubled for a 'typical' image). Next, we switched to twenty evenly spaced nodes were used to represent color spectra. Because this had a high storage cost (especially for images using zonal calculations), the four unevenly spaced node locations suggested by Meyer was used [72]. There was no qualitative loss in image quality observed going from the twenty to four samples, but this may say more about the arbitrary nature of most of the input spectral curves than about the quality of the spectral approximation. We suspect that filtering the input spectra before point sampling them will avoid most problems associated with only using a few sample locations.

Another basic utility class is the ray class, represented by a point of origin, and a unit-vector representing direction. The ray class has no allowed operations, but does have the basic 'method' (member function) that finds the point a certain distance along the ray. We have found that it is helpful to associate two other characteristics with a ray. The first is a material id which stores the material the ray is in (e.g. 'this ray is now traveling through glass'). The second characteristic is the attenuation of the ray. This makes the bookkeeping associated with adaptive ray tree pruning [44] straightforward.

A utility class that has been surprisingly useful is an orthonormal basis of three vectors. Though any two of these vectors uniquely defines the basis, all three are explicitly stored, trading space for execution time. These bases are used in viewing calculations, and to assign a local coordinate system to a surface. The basis is defined by three orthonormal vectors, $\vec{\mathbf{u}}$, $\vec{\mathbf{v}}$, and $\vec{\mathbf{w}}$.

Another utility class is the (u, v) pair. This class is useful for pixel sampling and texture mapping. The texture class itself adds another useful utility. In this implementation, the texture abstract class takes both a point in 3D, and a (u, v) pair. Surface textures and solid textures [84] are both subclasses of the texture class. A solid texture will use the point for texture generation, and the surface texture will use the (u, v) coordinate for color lookup. This 'send all information, needed or not' strategy can be inefficient, but it is a simple way to guarantee that the needed information is passed to a texture module.

A very useful utility class is a solid noise generator. We have implemented the generator given in Perlin's 1989 paper [85], and found it to be quite mechanical to implement. While it

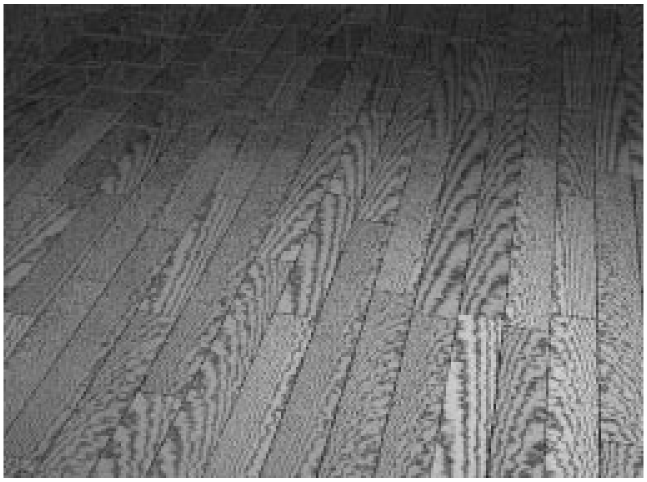

Figure 8.1: A board floor generated from a procedural solid texture.

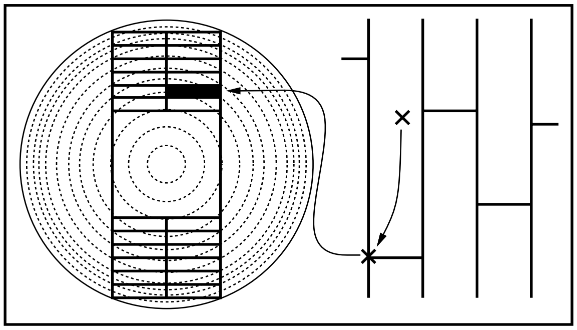

Figure 8.2: Use one corner of o board to find a solid noise seed for an index into the tree.

has been demonstrated that solid noise is useful in generating uniform textures such as marble, we have found that these techniques are too simple for semistructured patterns such as board floors, brick walls, and carpeted floors. Each element of these patterns (e.g. a specific brick) can be modeled using various transformations on solid noise. This needs to be added one or more randomly or semi-randomly varying parameters to make the elements vary among themselves.

As an example, a single board has a random grain pattern associated with a planer surface passing through a tree. The woodgrain of a tree is modeled in a manner similar to Perlin's marble: the basic structure of the tree is light and dark concentric rings, where the area (not distance) between rings is roughly constant. The radius of a given ring will decrease slightly as it moves up the tree (it is visually important that the rings are not perfect cylinders). The noise is used to add some irregularity to the geometry of the rings. The particular location of a board in a tree is what gives its unique character. Since the tree is of finite volume there are a finite set of specific boards that come form a tree, and each can be given a specific integer id.

We create our board texture by putting down an algorithmic (predefined) board pattern, and associating a specific board id to each board. This is easily accomplished by using the solid noise at a particular corner of the board as a seed to generate the board location in the tree (Figure 8.2). This guarantees that all points on a board are mapped to the same id in the tree. This procedure gives some irregularity to what types of boards (fine-grained versus loose-grained) are adjacent in the pattern. To ensure that there is no visible correlation between the ids of adjacent boards, we scale the corner points by some large factor before calling the solid noise function. An example floor generated by this method is shown in Figure 8.1.

This same basic idea can be used to generate bricks. Solid noise evaluated at one corner of a brick will give a seed to control some global parameter for that brick. If more than one parameter is needed, the other corners (or midpoints, etc.) can be also used for solid noise seed points. To generate the basic colors of wood or brick, we attempt to use physical spectral curves. Unfortunately this data can be hard to come by, and directly specifying spectral curves is not intuitive. In these cases we generate curves by 'mixing' standard artist's pigments, the curves for which can be found in [7]. When we have existing rgb data, we use Glassner's conversion method [39]. It has been our experience that using smooth curves for this conversion is highly preferable to using impulses.

8.2 Ray Tracing Primitives

As discussed in Chapters 5 and 6, finding the object first seen along a ray is a fundamental operation. Paul Heckbert has noted that this operation should be associated with an abstract 'geometrical object' class[51]. Specific objects, such as spheres and polygons, should be subclasses of the abstract class. This allows the addition of new geometrical object types without muddying existing code that uses ray tracing. Arvo and Kirk have generalized this idea by making the spatial ray intersection structures, such as Glassner's octree[38], themselves subclasses of geometrical object[65]. This makes sense because the program modules that use the ray tracing primitives do not need to know about underlying representations.

One useful thing that Arvo and Kirk have emphasized is that having the geometrical primitives (spheres, polygons, etc.) and the collection structures (octrees, lists, etc) be subclasses of the same abstract class, is that storage structures can be nested. This allows a particular storage class (a bounding volume hierarchy, for example), to contain, as members, other storage classes (octrees, regular grids, simple lists). Intelligent use of this generality can reduce running time. Unfortunately, just how to attain 'intelligent use' is not obvious.

Another useful characteristic of this class structure is that data hiding is quite natural. One of the basic primitives in the program is the triangle. A connected set of triangles forms a new 'mesh' primitive. If an octree data structure were used to store the primitives at a global level, it would be natural to want to add a mesh as a member of the octree. At appropriate times, the octree can query the mesh on whether a ray intersects the mesh. The mesh can then call on its list of triangles to decide whether the ray hits it. If there were many triangles, it would be essential that the mesh maintain its own internal efficiency structure (perhaps an octree). The general class structure allows such nesting, and is more modularized than it would be if the global octree were allowed access to the mesh's list of triangles.

This general grouping of spheres, polygons, octrees, etc. emphasizes an important point: classes with common access (member) functions should be subclasses of the same abstract class, even if their underlying representations are very different. Classes with different access rules, even if their underlying representations are very different, should not be grouped together, as seen with points and vectors in the last section.

8.3 Ray-Material Interaction

Several researchers have noted that reflection behavior should be encapsulated as one unit of a rendering system [66, 115, 47]. We treat light-material interaction as a component, where the reflection behavior is determined strictly from a set of material parameters. Traditionally this might be accomplished with one shading model with parameters including ks, kt, and kd [43]. One problem with such an approach is that physically implausible parameter combinations can be chosen by the user (e.g. kd = ks = 0, kt = 1). Implausible combinations may be useful for many applications, but if realism is desired, it is better to limit the user's choices.

We have used the idea that materials can be classed as families, each grouped by the parameters that effect their behavior. This way the user only needs to choose the relevant parameters for a particular material. Once the material is chosen it is treated as a black box component that responds to a limited protocol (much like geometrical objects' for ray-object intersection). The first way in which a material can be queried is by asking, 'given an incoming ray r, a point p on the surface, and a surface normal $\vec{\mathbf{n}}$, what rays r_i are reflected/transmitted by the material, and what is the attenuation k_i for each ray'. This will allow us to handle building the ray propagation code. For other lighting calculations, such as the direct lighting component, we need to query a material, 'what is your radiance, $L(\vec{\mathbf{v}}_{out})$, that comes from a source of radiance $L(\vec{\mathbf{v}}_{in})$ that subtends a solid angle ω?'. We also need to ask a material if it is a luminaire (source of light), and if so, how much light it emits in a particular direction.

The materials that we have implemented are:

conductor: Parameters n (refractive index), k (extinction coefficient), e (phong-style exponent). Example: aluminum.

dielectric: Parameters n (refractive index), a (filter coefficient), e (phong-style exponent). Example: glass.

lambertian: Parameter k_d (diffuse coefficient). Example: matte paint.

polished: Parameters k_d (diffuse coefficient of substrate), n (refractive index of polish), e (phong-style exponent). Example: gloss paint.

translucent: Parameters k_d^1 (diffuse coefficient of first side), k_d^2 (diffuse coefficient of first side), kt (transmission coefficient). Example: lampshade.

luminaire: Parameters k_d (diffuse coefficient), e (phong-style exponent). Example: light bulb.

These basic materials can be extended, but they have proven to be fairly good approximations to common real world materials. Conductors are sometimes a little difficult because the parameters n and k are‘ not intuitively controllable. We have found most data for conductors in [82]. The behavior of both conductors and dielectrics is determined using the Fresnel Equations, the full form of which can be found in [102, 98]. The polished surface is an approximation to a diffuse substrate with a thin dielectric covering. This means that for a given direction we first calculate the specular reflectivity $k_s(\theta)$, and then the remaining light is reflected diffusely, giving a diffuse reflectance of $(1 - k_s(\theta))k_d$. This allows glare effects to be approximated accurately. The phong-style exponent e is used to allow some spread in the reflected component of conductors, dielectrics, and polished materials. For smooth surfaces e is set to a large number.

The translucent surface reflects light diffusely from either side, and also allows some light to be diffusely transmitted. The luminaire acts as a diffuse reflector, and also emits power in a phong-style distribution. Large exponents are used if spot lights are desired.

The ray reflection/transmission behavior can be summarized as:

conductor: Generate one reflected ray with attenuation determined by Fresnel Equations. Perturb ray stochasticly if $e \neq \infty$.

dielectric: Generate one reflected ray and one transmitted ray with attenuations determined by Fresnel Equations. Perturb both rays stochasticly if $e \neq \infty$.

lambertian/luminaire: Generate one reflected ray stochastically with a cosine distribution.

polished: Generate one reflected ray from the polish with attenuation determined by Fresnel Equations. Perturb ray stochasticly if $e \neq \infty$. Generate one reflected ray from the diffuse substrate stochastically with a cosine distribution.

translucent: Generate two rays stochastically, one reflected, one transmitted, each with a stochastic cosine distribution.

It is useful to be able to turn off reflections from a particular material. We allow this to be done when the material is initialized. A conventional Whitted-style ray tracer would turn off reflections for the lambertain, translucent, and luminaire surfaces, and would turn off reflections off the substrate (but not the polish) of the polished surfaces. To maintain some form of dependent sampling, such as uncorrelated jittering, the reflection protocol should also accept a canonical (u, v) pair , to be used as a basis for any probabilistic reflection that might occur.

8.4 Zonal Calculations

Several recent zonal (radiosity: global lighting information is stored at a finite set of 'zones') systems are based on progressive refinement techniques [22, 2]. The theoretical basis for such systems is straightforward to extract. If the progressive refinement is viewed as power transport simulation, which implies fairly direct non-diffuse zonal solutions [5, 100, 42, 99, 95]. These solutions are easy to construct if we view the zone as a black box which collects power carrying rays, and later emits a group of power carrying rays that represent reflected power accumulated since the previous emission step.

This abstraction underlying zonal calculations can be stated: zones should receive, accumulate, and send power, and the mechanics of how this happens should be hidden. This is accomplished by defining a zonal-data module. In addition, the module should, after the zonal calculations are completed, be able to provide the radiance of the patch when viewed from a certain direction.

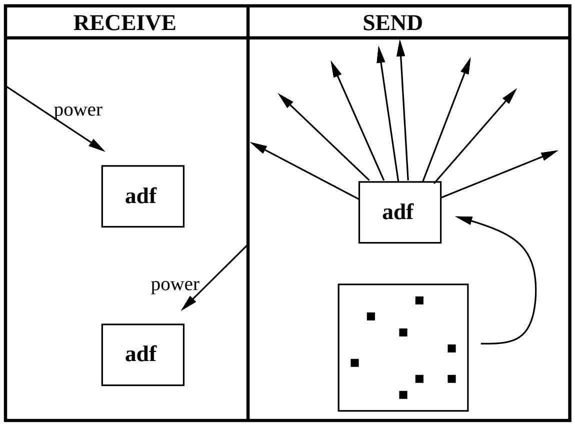

Figure 8.3: The two crucial methods of and adf (angular distribution function): receive power, and send power according to some set of sample points.

For a lambertian zone, the zonal-data module is easy to implement because there is no dependancy on the incoming direction of intensity. We need to store the total power, Φ, and the unsent accumulated power, Φ_u. Each new incoming ray carrying power Φ_i will imply $\Phi = \Phi + k_d\Phi_i$ and $\Phi_u = \Phi_u + k_d\Phi_i$. When it is time for the zone to emit, it will send N rays each carrying power Φ_u/N. These rays will be sent in a cosine distribution. Just as done with pixel sampling, we can derive N (u, v) pairs and then transform these to the appropriate (θ, ϕ) pairs. The radiance of the zone will just be $\Phi/(\pi A)$, where A is the area of the zone.

For a zone with directionally dependent reflection behavior, such as brushed steel, we must maintain the total and unsent power as some kind of directional table [42, 99, 95]. A simple way to do this is a simple spherical coordinate array of bins, with the total power going through each bin. The unsent and total power of the diffuse case must be generaized to a new black box, the angular distribution function (adf). This function maintains whatever information is necessary to receive power, and later send power as a set of rays (Figure 8.3).

The receiving method of a directionally dependent adf can be implemented by using the ray-material interaction module of Section 8.3. Simply reflect the incoming ray using the ray-material interaction module as a black box, and add the attenuated reflected power to whichever angular bin(s) the reflected ray(s) land in. The sending stage can be implemented using the sampling transformation methods, or by independantly sending a pattern of N rays from each angular bin. Because the adf actually stores spectral values, it can only be converted to a probability density by converting the entries to scalars. We use luminance to do this.

It would be very wasteful of storage to store an explicit directional table for diffuse surfaces, though it would still work. We therefore implement a lambertian adf by storing only the total and accumulated power. The black box interface still looks the same to the zonal module. To accomplish this in an extendable way, we add an access function to the ray-material interaction class which tells whether the reflection behavior is directionally dependent or independent, and whether the surface is reflective or reflective and transmissive. If the material is reflective and directionally independent (e.g. lambertian), it will use a adf module that internally only stores total and unsent power. If it is reflective and transmissive and directionally independent (e.g. translucent), then these quantities will be maintained both above and below the surface. If directionally dependent, the directional tables will be maintained either for $(0 < \theta < \pi/2)$ or $(0 < \theta < \pi)$ depending on whether the material is transmissive. If the material can provide an estimate of specularity (e.g. the phong exponent), then this can be used to choose the resolution of the table.

Once the adf modules have been initialized in each zonal module, their internal representation in invisible. This allows the programmer to detatch the local lighting models from the global light transport.

The zonal-data abstract class accomplishes two important functions. First, it removes any reference to surface reflection type from the light transport code. This makes the code more readable and allows new reflection types to be added in a modular fashion. The second important function is that it allows variable storage for the zonal-data of different reflection types. Thus, adding a surface with a large directional table does not force the lambertian surfaces to use more memory.

If the zone is textured, its average reflection properties must be found. To avoid aliasing problems, stochastic techniques are used to point sample the zone to estimate the average properties.

8.5 Viewing Calculations

There are two basic abstractions in the viewing calculations. These abstractions can apply to any ray tracing program, even if no zonal calculations are used. The first abstraction is to make the sampling distribution come from an abstract class. This lets the user flexibly choose and add sampling methods and filter functions in a natural way. This flexibility helped in the comparison tests of Sections 5.1 and 5.5. The second basic abstraction is to have the material (e.g. steel, glass, etc.) control the shading of a surface. Shading includes both responding to light (e.g. phong shading), and generating reflected or transmitted light. A similar abstraction was used by Arvo and Kirk[65]. This simplifies the shading calculations, and allows new material types and shading models to be added in a flexible manner.

One issue that remains unresolved is how to handle surfaces with complex material properties. For example, we could define a floor surface with alternating tiles made of marble and steel. The steel would respond to light as a metal, and the marble as a polished surface. It would also be desirable to be able to add a layer of dust (probably using a procedural texture), that would cover the marble and steel in a nonuniform manner, reducing the specularity of both. If the ‘material’ were to handle all shading in this situation, it would need access to steel, marble, and dust reflectance behavior, as well as the procedural texture describing the dust. This could be accomplished in a manner similar to a Renderman shade[115, 47], where the shading routine has access to the internals of reflection models and textures. Unfortunately, such a shader does not hide much information, and can become quite unwieldy. It would be very desirable to put the capabilities of a general shader in a class structure that preserves modularity and data hiding, but exactly how to create such a class structure is still a research topic.

8.6 Summary

Two basic programming strategies have been used to implement the code used for this work. The first was the creation of toolbox classes of points, vectors, colors, orthonormal bases, texture coordinates, and textures. These toolbox classes were used as primitive types, much like integers and floats in numerical codes, in the creation of the image generation code.

The second basic strategy was the creation of abstract classes of geometrical-object, sample-generator, zonal-data, and material-shader. These objects, practically speaking, replace explicit case statements with calls to virtual functions of the abstract classes. In the case of geometrical-objects, this allows the addition of new object types and search hierarchies. For sample-generators, this allows different sampling strategies and distributions to be plugged into the basic ray tracing module. The zonal-data class accomplishes data hiding on the accumulated power distributions, so efficient storage systems can be implemented for each surface type.

The least satisfying part of the implementation is the material-shader abstract class, which seems to need to much global information as currently implemented. It remains to be seen whether it is possible to have a more modular shading class structure while maintaining the power that can come with the global information.

CHAPTER 9

CONCLUSION

In this work I have attempted to put realistic image generation into a usable theoretical formulation, and provide an overview of some previous and new solution techniques for global illumination.

The basic physics of reflection and light transport were used to derive the rendering equation. The problem of generating an image was phrased in terms of evaluating the Global Radiance Function. The standard reflection models of computer graphics were reviewed, and modifications were made to the combined specular and diffuse reflection model to allow for Fresnel effects. This formulation of the rendering equation differs from previous formulations by explicitly accounting for transparent surfaces.

The physical rules governing reflection were also used to make improvements in reflection models. In diffuse transmission it was shown that light is filtered to the same extent regardless of which side of the surface the light comes from. This eliminates one of the parameters from previous diffuse transmission models. The microscopic structure of polished surfaces was used to justify coupling the diffuse and specular coefficients according to the Fresnel Equations. The Fresnel Equations are commonly used to vary the reflectivity of metal and transparent dielectrics, but have not been used before to vary the reflectivity of the polish and underlying diffuse substrate.

Image-based solution methods were phrased as a lazy evaluation of the Global Radiance Function; evaluation only took place for visible points. Several constraints were outlined for what part of the image function should contribute to each pixel, and a new separable, symmetric filter was developed that satisfies these constraints.

A stochastic shadow ray generation method was introduced that greatly reduces the number of shadow rays needed for scenes with multiple light sources. The sampling distributions used for shadow rays and other dimensions of the integral were evaluated by introducing to computer graphics the notion of discrepancy from numerical integration theory. The use of discrepancy provided some insight not given by the signal processing theory traditionally used in computer graphics. As part of this discussion a new sampling scheme, separate sampling, was introduced. Separate sampling was shown to be as efficient to generate as jittered sampling, while often outperforming Poisson disk sampling. It also can generate distributions for any positive integer number of samples, including primes.

The peculiarities of the sampling spaces used in distributed ray tracing were shown to preclude naive hierarchical sampling. It was demonstrated that hierarchical sampling can greatly reduce noise, however, if we have sufficient knowledge of the sampling space.

Zonal methods represent the opposite extreme of image methods, where all function values are computed and stored, and each evaluation is a table lookup. The zonal method was phrased as a transport simulation, similar to progressive refinement radiosity methods. Using this direct simulation model, it is straightforward to generate zonal methods for anisotropic reflection. This requires storing accumulated power in a directional table for each zone.

It was also shown that simulation allows for surfaces which are not zoned to interact with those that are. This is a generalization of the diffuse and specular ray tracing transport work of Malley. This technique can be useful for highly complex or difficult to zone surfaces such as a human face. For ray tracing methods, $O(N)$ rays are required for scenes with N zones that have a bounded area ratio.

The zonal solution methods can be applied to participating media in a fairly natural manner. This also applies to media with anisotropic scattering characteristics, but such a solution requires a large amount of storage.

Wavelength dependent solutions introduce some complications, but can be handled by traditional point sampling techniques. Time dependent solutions are easily handled by image-based solution methods, but are very difficult to apply using zonal methods.

In implementing a global renderer, it is possible encapsulate the local illumination code so that it is independent of the global illumination code. The standard techniques of object oriented programming are useful in accomplishing this.

In closing, there are many open problems that are worth looking into, of which a partial list follows:

- Is a sophisticated model of glossy reflection needed for Computer Graphics applications, or will simple empirical models suffice?
- Are anisotropically scattering media needed for Computer Graphics?
- How should indirect lighting in outdoor scenes be handled?
- How should interaction between indoor and outdoor scenes (at windows) be handled?
- How realistically must we model the characteristics of light sources?
- How should we restrict the allowed permutations of separately generated samples? Can discrepancy analysis be of use?
- What automatic gridding techniques could be used for zonal methods?
- How can discrepancy be generalized to non-rectangular sampling spaces?

- How should modeling interact with reflection models? E.g. should wet wood become darker automatically?
- Should discretization of a complex zonal environment be view dependent?
- What is a valid error metric for a computed image?
- For the purposes of complexity analysis, what is an 'average' scene?
- How do we determine the error of an approximate image?
- How can perceptual science guide us in displaying an image, so that it evokes the same response as a real scene, rather than that of of a photograph?
- Is there an algorithm that can efficiently calculate solutions for indoor, outdoor, and combined scenes?

APPENDIX A

RAY-SURFACE INTERSECTION CALCULATIONS

A.1 Spheres

Suppose we have a ray $r = \mathbf{o} + t\vec{\mathbf{v}}$ and a sphere S defined by a center point $\mathbf{c}$ and a radius R. The equation for the sphere is:

$$(x - c_x)^2 + (y - c_y)^2 + (z - c_z)^2 = R^2$$

Which can also be written in terms of points $\mathbf{p}$ that lie on the sphere:

$$(\mathbf{p} - \mathbf{c}) \cdot (\mathbf{p} - \mathbf{c}) = R^2$$

The ray r will hit the sphere S at any values of t that satisfy:

$$(\mathbf{o} + t\vec{\mathbf{v}} - \mathbf{c}) \cdot (\mathbf{o} + t\vec{\mathbf{v}} - \mathbf{c}) = R^2$$

Expanding this yields a quadratic in t:

$$\vec{\mathbf{v}} \cdot \vec{\mathbf{v}} t^2 + 2(\mathbf{o} - \mathbf{c}) \cdot \vec{\mathbf{v}} t + (\mathbf{o} - \mathbf{c}) \cdot (\mathbf{o} - \mathbf{c}) - R^2 = 0$$

Solving for t gives:

$$t = \frac{-4(\mathbf{o} - \mathbf{c}) \cdot \vec{\mathbf{v}} \pm \sqrt{4\left[(\mathbf{o} - \mathbf{c}) \cdot \vec{\mathbf{v}}\right]^2 - 4\vec{\mathbf{v}} \cdot \vec{\mathbf{v}}\left[(\mathbf{o} - \mathbf{c}) \cdot (\mathbf{o} - \mathbf{c}) - R^2\right]}}{2\vec{\mathbf{v}} \cdot \vec{\mathbf{v}}} \qquad \text{(A.1)}$$

If the quantity inside the square root is positive then there are two points where the ray r hits the sphere S. If the quantity is zero then r grazes S. If the quantity is negative then there are no real solutions and r misses S.

A.2 Triangles

Suppose we have a ray $r = \mathbf{o} + t\mathbf{v}$ and a triangle T is defined by its three non-colinear vertices $\mathbf{p}_0$, $\mathbf{p}_1$, and $\mathbf{p}_2$, and we want to find out whether r intersects T.

The plane P containing the triangle can be written as $\vec{\mathbf{N}} \cdot \mathbf{p} + D = 0$. Here $\vec{\mathbf{N}}$ is the surface normal of P. The normal $\vec{\mathbf{N}}$ can be found by taking the cross product of two vectors in the direction of two sides of T:

$$\vec{\mathbf{N}} = (\mathbf{p}_2 - \mathbf{p}_0) \times (\mathbf{p}_1 - \mathbf{p}_0)$$

The normal $\vec{\mathbf{N}}$ can then be scaled or flipped. Assuming $\vec{\mathbf{N}}$ is scaled to be of unit length (denoted by $\mathbf{N}$), D is:

$$D = -\mathbf{N} \cdot \mathbf{p}_i$$

where $\mathbf{p}_i$ is any of the vertices of T.

The ray r will hit the plane P at a distance t_0 satisfying:

$$\mathbf{N} \cdot (\mathbf{o} + t_0\mathbf{v}) + D = 0.$$

Solving for t_0 yields:

$$t_0 = -\frac{\mathbf{N} \cdot \mathbf{o} + D}{\mathbf{N} \cdot \mathbf{v}}$$

We can use the three vertices to set up a simple (u, v) coordinate system on the plane P containing the triangle. Any point $\mathbf{p}$ on the plane can be described in terms of this coordinate system as follows:

$$\mathbf{p} = \mathbf{p}_0 + u(\mathbf{p}_1 - \mathbf{p}_0) + v(\mathbf{p}_2 - \mathbf{p}_0) \tag{A.2}$$

The point $\mathbf{p} = \mathbf{o} + t_0\mathbf{v}$ is inside the triangle if and only if:

1. $0 \le u \le 1$.

2. $0 \le v \le 1$.

3. $(u + v) \le 1$.

So to find whether a ray hits a triangle T we can follow the steps:

1. Find the point $\mathbf{p}$ in plane P hit by ray r.

2. Find the (u, v) pair associated with $\mathbf{p}$.

3. Examine (u, v) to see if $\mathbf{p}$ is inside the triangle T.

APPENDIX B

MONTE CARLO METHODS

B.1 Introduction

In this appendix the basic Monte Carlo solution methods for definite integrals and sums are outlined. These techniques are then straightforwardly extended to certain integral and linear equations. All of the material of this appendix is also covered in several of the classic Monte Carlo texts. This appendix differs by being geared toward classes of problems that crop up in Computer Graphics. Readers interested in a broader treatment of Monte Carlo techniques should consult one of the classic Monte Carlo texts[46, 97, 45, 124].

B.2 Background

Before getting to the specifics of Monte Carlo techniques, we need several definitions, the most important of which are: *random variable*, *expected value*, and *variance*.

Loosely speaking, a *random variable* x is a scalar or vector quantity that 'randomly' takes on some value, and the behavior of x is entirely described by the distribution of values it takes. This distribution of values can be quantitatively described by the *probability density function*, f, associated with x (the relationship is denoted $x \sim f$). If x ranges over some region Ω, then the probability that x will take on a value in some subregion $\Omega_i \subset \Omega$ is given by the integral:

$$P(x \in \Omega_i) = \int_{x' \in \Omega_i} f(x') d\mu(x') \quad (f : \Omega \rightarrow \Re^1) \tag{B.1}$$

Here $P(event)$ is the probability that $event$ is true, so the integral is the probability that x takes on a value in the region Ω_i. The measure μ is the measure on our probability space. In graphics Ω is typically an area ($d\mu = dA = dxdy$), or a set of directions (points on a unit sphere: $d\mu = d\omega = \sin\theta d\theta d\phi$). The density f has two characteristics:

$$f(x) \geq 0 \quad \text{(Probability is nonnegative)} \tag{B.2}$$

$$\int_{x' \in \Omega} f(x') d\mu(x') = 1 \quad (x \text{ has a value in } \Omega) \tag{B.3}$$

As an example, the canonical random variable ξ takes on values between zero and one with uniform probability. This implies that:

$$f(\xi) = \begin{cases} 1 & \text{if } 0 \leq \xi \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

The probability that ξ takes on a value in a certain region is:

$$P(a < \xi < b) = \int_a^b 1 d\xi' = b - a$$

The average value a random variable will take on is called its *expected value*, $E(x)$:

$$E(x) = \int_{x' \in \Omega} x' f(x') d\mu(x')$$

The expected value has a surprising and useful property: *the expected value of the sum of two random variables is the sum of the expected values of those variables* ($E(x + y) = E(x) + E(y)$). This property holds whether or not the variables are independent! Since the sum of two random variables is itself a random variable, this principle generalizes. Since a function of x is itself a random variable, we can write down the expected value of a function $g(x)$:

$$E(g(x)) = \int_{x' \in \Omega} g(x')f(x')d\mu(x')$$

The *variance*, $var(x)$, of a random variable is the expected value of the square of the difference between x and $E(x)$:

$$var(x) = E([x - E(x)]^2) = E(x^2) - [E(x)]^2$$

The variance of a sum of random variables is the sum of the variances *if the variables are independent.* The square root of the variance is called the *standard deviation*, which gives some indication of absolute deviation from the expected value.

Many problems involve sums of independent random variables x_i, where the variables share a common density f. Such variables are said to be *independent identically distributed* random variables. When the sum is divided by the number of variables, we get an estimate of $E(x)$:

$$E(x) \approx \frac{1}{N} \sum_{i=1}^{N} x_i$$

This idea can be generalized to the *Law of Large Numbers*:

$$P\left[E(x) = \lim_{N \to \infty} \frac{1}{N} \sum_{i=1}^{N} x_i\right] = 1$$

This idea leads naturally to the idea of Monte Carlo estimation of integrals.

B.3 Monte Carlo Integration

From the last section we saw that for a function g and a random variable $x \sim f$, we can approximate the expected value of $g(x)$ by a sum:

$$E(g(x)) = \int_{x' \in \Omega} g(x')f(x')d\mu(x') \approx \frac{1}{N} \sum_{i=1}^{N} g(x_i) \qquad \text{(B.4)}$$

Because the expected value can be expressed as an integral, the integral is also approximated by the sum. The form of Equation B.4 is a bit awkward; we would usually like to approximate an integral of a single function h rather than a product gf. We can get around this by substituting $h = gf$ as the integrand:

$$\int_{x' \in \Omega} h(x') d\mu(x') \approx \frac{1}{N} \sum_{i=1}^{N} \frac{h(x_i)}{f(x_i)} \tag{B.5}$$

For this formula to be valid, f must be positive where h is nonzero.

Variance can be used to measure the reliability of the estimate. Both estimates are *unbiased*, which means that the expected values are what we would expect. The simple term $h(x_i)/f(x_i)$ is called the *primary estimator*, and the average of many primary estimators is a *secondary estimator*. The secondary estimator is preferred because its variance is lower. The variance of the estimate is:

$$var\left(\frac{1}{N} \sum_{i=1}^{N} \frac{h(x_i)}{f(x_i)}\right) = \frac{var(\frac{h(x)}{f(x)})}{N} \tag{B.6}$$

So to get a good estimate, we want as many samples as possible, and we want the density h/f to have a low variance (similar shape). Choosing f intelligently is called importance sampling, because if f is large where h is large, there will be more samples in important regions. Equation B.4 also shows the fundamental problem with Monte Carlo integration: *diminishing return*. Because the variance of the estimate is proportional to $1/N$, the standard deviation is proportional to $1/\sqrt{N}$. Since the error in the estimate behaves similarly to the standard deviation, we will need to quadruple N to halve the error.

Another way to reduce variance is to partition Ω, the domain of the integral, into several smaller domains Ω_i, and evaluate the integral as a sum of integrals over the Ω_i. This is called stratified sampling. Normally only one sample is taken in each Ω_i (with density f_i), and in this

method	*sampling function*	*variance*	*samples needed for standard error of 0.008*
importance	$(6-x)/(16)$	$56.8N^{-1}$	887,500
importance	$1/4$	$21.3N^{-1}$	332,812
importance	$(x+2)/16$	$6.3N^{-1}$	98,437
importance	$x/8$	0	1
stratified	uniform	$21.3N^{-3}$	70

Table B.1: Variance for Monte Carlo Estimate of $\int_0^4 x\,dx$

case the variance of the estimate is:

$$var\left(\sum_{i=1}^{N}\frac{h(x_i)}{f_i(x_i)}\right)=\sum_{i=1}^{N} var\left(\frac{h(x_i)}{f_i(x_i)}\right) \tag{B.7}$$

As an example of the Monte Carlo solution of an integral I set $h(x)$ to be x over the interval (0, 4):

$$I=\int_0^4 x\,dx=8 \tag{B.8}$$

The great impact of the shape of the function f on the variance of the N sample estimate is shown in Table B.1. Note that the variance is lessened when the shape of f is similar to the shape of h. The variance drops to zero if $p = Ch$, but h is not usually known or we would not have to resort to Monte Carlo. One important principle illustrated in Table B.1 is that stratified sampling is often *far* superior to importance sampling. Although the variance for this stratification on I is inversely proportional to the cube of the number of samples, there is no general result for the behavior of variance under stratification. There are some functions where stratification does no good. An example is a white noise function, where the variance is constant for all regions. A poorly chosen stratification can even increase the variance for some functions.

B.4 Monte Carlo Solution to an Integral Equation

The ‘rendering equation’ is a *Fredholm Equation of the Second Kind.* Such equations have the form:

$$a(x) = b(x) + \int_{x' \in \Omega} k(x, x')a(x')d\mu(x')$$

Where b and k are known. To apply the equipment of the last section we can repeatedly substitute a into the integral:

$$\begin{aligned} a(x) &= b(x) + \\ & \int_{x' \in \Omega} k(x, x')b(x')d\mu(x') + \\ & \int_{(x', x'') \in \Omega^2} k(x, x')k(x', x'')b(x'')d\mu(x')d\mu(x'') + \\ & \int_{(x', x'', x''') \in \Omega^3} k(x, x')k(x', x'')k(x'', x''')b(x''')d\mu(x')d\mu(x'')d\mu(x''') + \cdots \end{aligned} \tag{B.9}$$

A primary estimator for the first integral in the series is:

$$\frac{k(x, x')b(x')}{f_1(x')} \quad : x' \sim f_1 \tag{B.10}$$

A primary estimator for the second integral is:

$$\frac{k(x, x')k(x', x'')b(x'')}{f_2(x', x'')} \quad : (x', x'') \sim f_2 \tag{B.11}$$

And the third integral:

$$\frac{k(x, x')k(x', x'')k(x'', x''')b(x''')}{f_3(x', x'', x''')} \quad : (x', x'', x''') \sim f_3 \tag{B.12}$$

We could simple estimate each integral separately and add these estimates to form a estimate for the truncated series. What is usually done, however, is to reuse the share sample points between integrals. To do this, we choose a chain of samples $(x^1, x^2, x^3, \cdots, x^n)$. The estimator for the first n integrals in Equation B.9 is:

$$\begin{aligned} a(x) &= b(x) + \\ &\frac{k(x,x^1)b(x^1)}{f_1(x^1)} + \\ &\frac{k(x,x^1)k(x^1x^2)b(x^2)}{f_2(x^1,x^2)} + \\ &\frac{k(x,x^1)k(x^1,x^2)k(x^2,x^3)b(x^3)}{f_3(x^1,x^2,x^3)} + \\ &\vdots \\ &\frac{k(x,x^1)\cdots k(x^{n-1},x^n)b(x^n)}{f_n(x^1,\cdots,x^n)} \end{aligned} \tag{B.13}$$

We should probably have some misgivings about reusing the sample points for each integral, but this will not bias our sample because, as stated earlier, the expected value of a sum is the sum of expected values, even if the variable being summed are *not* independent.

B.5 Monte Carlo Estimates of Sums and Linear Equations

The techniques used to estimate integrals can also be used to estimate sums. One way to see this is to consider a sum to be a special case of an integral (and discrete probability a special case of continuous probability). A primary estimator for a sum is:

$$\sum_{i=1}^{N} h_i \approx \frac{h_i}{f(i)}$$

Here $f(i)$ is the probability of choosing h_i. As with integrals, a lower variance secondary estimator can be developed by averaging multiple instances of the primary estimator.

Just as Monte Carlo integration extends to integral equations, Monte Carlo summation extends to linear equations. Consider the linear system:

$$x = b + Ax$$

where x is an unknown column vector of length N, b is a known column vector of length N, and A is a known $N \times N$ matrix. As with integrals, we first expand the equation into a series:

$$x = b + Ab + A^2b + A^3b + \cdots$$

Any particular element of x is given by:

$$\begin{aligned} x_i &= b_i + \\ & \sum_{j=1}^{N} A_{ij}b_j + \\ & \sum_{j=1,k=1}^{N} A_{ij}A_{jk}b_k + \\ & \sum_{j=1,k=1,l=1}^{N} A_{ij}A_{jk}A_{kl}b_l + \cdots \end{aligned} \tag{B.14}$$

As with the integral equations, we can generate a series $(j, k, l, m, \cdots)$, and generate a primary estimator for x_i:

$$\begin{aligned} x_i &\approx b_i + \\ & \frac{A_{ij}b_j}{f_1(j)} + \\ & \frac{A_{ij}A_{jk}b_k}{f_2(j,k)} + \\ & \frac{A_{ij}A_{jk}A_{kl}b_l}{f_3(j,k,l)} + \cdots \end{aligned} \tag{B.15}$$

B.6 Generating Random Numbers With Non-Uniform Densities

For Monte Carlo methods we must often generate random points according to some probability density function, or random rays according to a directional probability density. In this section a method for one and two dimensional random variables is described. The discussion closely follows that of Shreider[97].

If the density is a one dimensional $f(x)$ defined over the interval $x \in [a, b]$, then we can generate random numbers α_i that have density f from a set of uniform random numbers ξ_i, where $\xi_i \in [0, 1]$. To do this we need the probability distribution function $F(x)$:

$$F(x) = \int_a^x f(x') d\mu(x') \tag{B.16}$$

To get α_i we simply transform ξ_i:

$$\alpha_i = F^{-1}(\xi_i) \tag{B.17}$$

where F^{-1} is the inverse of F. If F is not analytically invertable then numerical methods will suffice because an inverse exists for all valid probability distribution functions.

If we have a two dimensional density (x, y) defined on $[a, b : c, d]$ then we need the two dimensional distribution function:

$$F(x, y) = \int_c^y \int_a^x f(x', y') d\mu(x', y') \tag{B.18}$$

We first choose an x_i using the marginal distribution $F(x, d)$, and then choose y_i according to $F(x_i, y)/F(x_i, d)$. If $f(x, y)$ is separable (expressable as $g(x)h(y)$), then the one dimensional techniques can be used on each dimension.

To choose reflected ray directions for zonal calculations or distributed ray tracing, we can think of the problem as choosing points on the unit sphere or hemisphere (since each ray direction ψ can be expressed as a point on the sphere). For example, suppose that we want to choose rays according to the density:

$$p(\theta, \phi) = \frac{n+1}{2\pi} \cos^n \theta \qquad \text{(B.19)}$$

Where n is a Phong-like exponent, θ is the angle from the surface normal and $\theta \in [0, \pi/2]$ (is on the upper hemisphere) and ϕ is the azimuthal angle ($\phi \in [0, 2\pi]$). The distribution function is:

$$P(\theta, \phi) = \int_0^{\phi} \int_0^{\theta} p(\theta', \phi') \sin \theta' d\theta' d\phi' \qquad \text{(B.20)}$$

The $\cos \theta'$ term arises because on the sphere $d\omega = \cos \theta d\theta d\phi$. When the marginal densities are found, p (as expected) is separable and we find that a (r_1, r_2) pair of uniform random numbers can be transformed to a direction by:

$$(\theta, \phi) = (\arccos((1 - r_1)^{\frac{1}{n+1}}), 2\pi r_2) \qquad \text{(B.21)}$$

One nice thing about this method is that a set of jittered points on the unit square can be easily transformed to a set of jittered points on the hemisphere with a distribution of Equation B.19. If n is set to 1 then we have a diffuse distribution needed for a Monte Carlo zonal method.

Other example results are: to choose points uniformly from a disk of radius R, apply the transformation $\theta = 2\pi r_1$, $r = R\sqrt{r_2}$. To choose random points on a triangle defined by vertices p_0, p_1, and p_2, apply the transformation: $a = 1 - \sqrt{1 - r_1}$, $b = (1 - a)r_2$, and the random point p will is: $p = p_0 + a(p_1 - p_0) + b(p_2 - p_0)$.

BIBLIOGRAPHY

[1] John M. Airey and Ming Ouh-young. Two adaptive techniques let progressive radiosity outperform the traditional radiosity algorithm. Technical Report TR89-20, University of North Carolina at Chapel Hill, August 1989.

[2] John M. Airey, John H. Rohlf, , and Frederick P. Brooks. Towards image realism with interactive update rates in complex virtual building environments. *Computer Graphics*, 24(1):41–50, 1990. ACM Workshop on Interactive Graphics Proceedings.

[3] John Amanatides. Ray tracing with cones. *Computer Graphics*, 18(3):129–135, July 1984. ACM Siggraph '84 Conference Proceedings.

[4] John Amanatides and Don P. Mitchell. Antialiasing of interlaced video animation. *Computer Graphics*, 24(3):77–86, August 1990. ACM Siggraph '90 Conference Proceedings.

[5] James Arvo. Backward ray tracing. *Developments in Ray Tracing*, pages 259–263, 1985. ACM Siggraph '85 Course Notes.

[6] James Arvo and David Kirk. Particle transport and image synthesis. *Computer Graphics*, 24(3):63–66, August 1990. ACM Siggraph '90 Conference Proceedings.

[7] Norman F. Barnes. Color characteristics of artists' pigments. *Journal of the Optical Society of America*, May 1939.

[8] Daniel R. Baum, Holly E. Rushmeier, and James M. Winget. Improving radiosity solutions through the use of analytically determined form-factors. *Computer Graphics*, 23(3):325–334, July 1989. ACM Siggraph '89 Conference Proceedings.

[9] Daniel R. Baum, John R. Wallace, Michael F. Cohen, and Donald P. Greenberg. The back-buffer algorithm: an extension of the radiosity method to dynamic environments. *Visual Computer*, 2:325–334, February 1986.

[10] James Blinn. Simulation of wrinkled surfaces. *Computer Graphics*, 12(3):286–292, August 1978. ACM Siggraph '78 Conference Proceedings.

[11] James F. Blinn. Light reflection functions for simulation of clouds and dusty surfaces. *Computer Graphics*, 16(3):21–30, July 1982. ACM Siggraph '82 Conference Proceedings.

[12] James F. Blinn. Return of the jagg. *IEEE Computer Graphics and Applications*, 9(2):82–89, 1989.

[13] James F. Blinn. What we need around here is more aliasing. *IEEE Computer Graphics and Applications*, 9(1):75–79, 1989.

[14] C. Bouville, J. L. Dubois, I. Marchal, and M. L. Viaud. Monte-carlo integration applied to an illumination model. In *Eurographics '88*, pages 483–497, 1988.

[15] William E. Brackett, Wayne L. Fink, and William Pierpoint. Interior point-by-point calculations in obstructed spaces. *Journal of the Illumination Engineering Society*, pages 14–25, October 1983.

[16] Brian Cabral, Nelson Max, and Rebecca Springmeyer. Bidirectional reflectance functions from surface bump maps. *Computer Graphics*, 21(4):273–282, July 1987. ACM Siggraph '87 Conference Proceedings.

[17] A. T. Campbell and Donald S. Fussell. Adaptive mesh generation fo global diffuse illumination. *Computer Graphics*, 24(3):155–164, August 1990. ACM Siggraph '90 Conference Proceedings.

[18] Hong Chen and En-Hau Wu. An efficient radiosity solution for bump texture generation. *Computer Graphics*, 24(3):125–134, August 1990. ACM Siggraph '90 Conference Proceedings.

[19] Shenchang Eric Chen. Incremental radiosity: An extension of progressive radiosity to an interactive image synthesis system. *Computer Graphics*, 24(3):135–144, August 1990. ACM Siggraph '90 Conference Proceedings.

[20] Yong C. Chen. Lens effect on synthetic image generation based on light particle theory. *Visual Computer*, 3:125–136, 1987.

[21] F. J. J. Clarke and D. J. Parry. Helmholtz reciprocity: Its validity and application to reflectometry. *Lighting Research and Technology*, 17(1):1–11, 1985.

[22] Michael F. Cohen, Shenchang Eric Chen, John R. Wallace, and Donald P. Greenberg. A progressive refinement approach to fast radiosity image generation. *Computer Graphics*, 22(4):75–84, August 1988. ACM Siggraph '88 Conference Proceedings.

[23] Michael F. Cohen and Donald P. Greenberg. The hemi-cube: a radiosity solution for complex environments. *Computer Graphics*, 19(3):31–40, July 1985. ACM Siggraph '85 Conference Proceedings.

[24] Micheal F. Cohen, Donald P. Greenberg, David S. Immel, and Philip J. Brock. An efficient radioisty approach for realistic image synthesis. *IEEE Computer Graphics and Applications*, 6(2):26–35, 1986.

[25] Robert L. Cook. Stochastic sampling in computer graphics. *ACM Transactions on Graphics*, 5(1):51–72, January 1986.

[26] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. *Computer Graphics*, 18(4):165–174, July 1984. ACM Siggraph '84 Conference Proceedings.

[27] Robert L. Cook and Kennneth E. Torrance. A reflectance model for computer graphics. *Computer Graphics*, 15(3):307–316, August 1981. ACM Siggraph '81 Conference Proceedings.

[28] R. C. Corlett. Direct monte carlo calculation of radiative heat transfer in vacuum. *Journal of Heat Transfer*, pages 376–382, November 1966.

[29] O. Devillers. Tools to study the efficiency of space subdivision structures for ray tracing. In *Second Annual Conference on Computer Graphics in Paris*, pages 467–481, September 1989.

[30] Mark A. Z. Dippe and Erling Henry Wold. Antialiasing through stochastic sampling. *Computer Graphics*, 19(3):69–78, July 1985. ACM Siggraph '85 Conference Proceedings.

[31] Walter G. Driscoll. *Handbook of Optics.* McGraw-Hill, New York, N.Y., 1978.

[32] Mark e. Lee, Richard A. Redner, and Samuel P. Uselton. Statistically optimized sampling for distributed ray tracing. *Computer Graphics*, 19(3):61–68, July 1985. ACM Siggraph '85 Conference Proceedings.

[33] David S. Falk, Dieter R. Brill, and David G. Stork. *Seeing the Light: Optics in Nature, Photography, Color, Vision, and Holography.* Harper and Row, New York, N.Y., 1986.

[34] Robert L. Feller, editor. *Artists' Pigments: A Handbook of their History and Characteristics.* Cambridge University Press, London., 1986.

[35] Eugene L. Fiume. *The Mathematical Structure of Raster Graphics.* Academic Press, San Diego, CA, 1989.

[36] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice.* Addison-Wesley, Reading, MA, second edition, 1990.

[37] Alain Fournier and Eugene Fiume. Constant-time filtering with space-variant kernels. *Computer Graphics*, 22(4):229–238, August 1988. ACM Siggraph '88 Conference Proceedings.

[38] Andrew S. Glassner. Space subdivision for fast ray tracing. *IEEE Computer Graphics and Applications*, 4(10):15–22, 1984.

[39] Andrew S. Glassner. How to derive a spectrum from an rgb triplet. *IEEE Computer Graphics and Applications*, 9(7):95–99, 1989.

[40] Cindy M. Goral, Kenneth E. Torrance, and Donald P. Greenberg. Modeling the interaction of light between diffuse surfaces. *Computer Graphics*, 18(4):213–222, July 1984. ACM Siggraph '84 Conference Proceedings.

[41] Paul Haeberli. The accumulation buffer: Hardware support for high-quality rendering. *Computer Graphics*, 24(3):309–318, August 1990. ACM Siggraph '90 Conference Proceedings.

[42] David Edward Hall. An analysis and modification of shao's radiosity method for computer graphics image synthesis. Master's thesis, Department of Mechanical Engineering, Georgia Institute of Technology, March 1990.

[43] Roy Hall. *Illumination and Color in Computer Generated Imagery.* Springer-Verlag, New York, N.Y., 1988.

[44] Roy Hall and Donald P. Greenberg. A testbed for realistic image synthesis. *IEEE Computer Graphics and Applications*, 3(8):10–20, 1983.

[45] John H. Halton. A retrospective and prospective of the monte carlo method. *SIAM Review*, 12(1):1–63, January 1970.

[46] J. M. Hammerley and D. C. Handscomb. *Monte Carlo Methods.* Wiley, New York, N.Y., 1964.

[47] Pat Hanrahan and Jim Lawson. A language for shading and lighting calculations. *Computer Graphics*, 24(3):289–298, August 1990. ACM Siggraph '90 Conference Proceedings.

[48] Pat Hanrahan and David Salzman. A rapid hierarchical radiosity algorithm for unoccluded environments. In *Proceedings of the Eurographics Workshop on Photosimulation, Realism and Physics in Computer Graphics*, pages 151–171, June 1990.

[49] Eugene Hecht and Alfred Zajac. *Optics.* Addison-Wesley, Reading, MA, 1974.

[50] Paul S. Heckbert. Filtering by repeated integration. *Computer Graphics*, 20(4):315–321, August 1986. ACM Siggraph '86 Conference Proceedings.

[51] Paul S. Heckbert. Fundamentals of texture mapping and image warping. Master's thesis, Department of E.E. and C.S., Georgia Institute of Technology, June 1989.

[52] Paul S. Heckbert. Writing a ray tracer. In Andrew S. Glassner, editor, *An Introduction to Ray Tracing.* Academic Press, San Diego, CA, 1989.

[53] Paul S. Heckbert. Adaptive radiosity textures for bidirectional ray tracing. *Computer Graphics*, 24(3):145–154, August 1990. ACM Siggraph '90 Conference Proceedings.

[54] J. R. Howell. Thermal radiation in participating media: The past, the present, and some possible futures. *Journal of Heat Transfer*, 110:1220–1227, November 1988.

[55] J. R. Howell and M. Perlmutter. Monte carlo solution of thermal transfer through radiant media between gray walls. *Journal of Heat Transfer*, pages 116–122, February 1964.

[56] David S. Immel, Michael F. Cohen, and Donald P. Greenberg. A radiosity method for non-diffuse environments. *Computer Graphics*, 20(4):133–142, August 1986. ACM Siggraph '86 Conference Proceedings.

[57] American National Standard Institute. Nomenclature and definitions for illumination engineering. ANSI Report, 1986. ANSI/IES RP-16-1986.

[58] Theodore M. Jenkins, Walter R. Nelson, and Alessandro Rindi, editors. *Monte Carlo Transport of Electrons and Photons.* Plenum Press, New York, N.Y., 1988.

[59] J. Kajiya and M. Ullner. Filtering high quality text for display on raster scan devices. *Computer Graphics*, 15(3):7–15, August 1981. ACM Siggraph '81 Conference Proceedings.

[60] James T. Kajiya. Anisotropic reflection models. *Computer Graphics*, 19(3):15–22, July 1985. ACM Siggraph '85 Conference Proceedings.

[61] James T. Kajiya. The rendering equation. *Computer Graphics*, 20(4):143–150, August 1986. ACM Siggraph '86 Conference Proceedings.

[62] James T. Kajiya and B. P. Von Herzen. Ray tracing volume densities. *Computer Graphics*, 18(4):165–174, July 1984. ACM Siggraph '84 Conference Proceedings.

[63] Malvin H. Kalos and Paula A. Whitlock. *Monte Carlo Methods*. John Wiley and Sons, New York, N.Y., 1986.

[64] Douglas Scott Kay. Transparancy, refraction and ray tracing for computer synthesized images. Master's thesis, Cornell University, January 1979.

[65] David Kirk and James Arvo. The ray tracing kernal. In *Proceedins of Ausgraph*, pages 75–82, July 1988.

[66] David Kirk and James Arvo. The ray tracing kernal. In *Proceedins of Ausgraph*, pages 75–82, July 1988.

[67] R. Victor Klassen. Modeling the effect of the atmosphere on light. *ACM Transactions on Graphics*, 6(3):215–237, July 1987.

[68] Mark Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3):29–37, 1988.

[69] Thomas J. V. Malley. A shading method for computer generated images. Master's thesis, University of Utah, June 1988.

[70] Nelson L. Max. Antialiasing scan-line data. *IEEE Computer Graphics and Applications*, 10(1):18–30, January 1990.

[71] Gary W. Meyer. Wavelength selection for synthetic image generation. *Computer Vision, Graphics, and Image Processing*, 41:57–79, 1988.

[72] Gary W. Meyer, Holly E. Rushmeyer, Michael F. Cohen, Donald P. Greenberg, and Kenneth E. Torrance. An experimental evaluation of computer graphics imagery. *ACM Transactions on Graphics*, 5(1):30–50, January 1986.

[73] W. E. Knowles Middleton and A. G. Mungall. The luminous directional reflectance of snow. *Journal of the Optical Society of America*, 42(8):572–579, August 1952.

[74] Don P. Mitchell and Arun N. Netravali. Reconstruction filters in computer graphics. *Computer Graphics*, 22(4):221–228, August 1988. ACM Siggraph '88 Conference Proceedings.

[75] Hans P. Moravec. 3d graphics and the wave theory. *Computer Graphics*, 15(3):289–296, August 1981. ACM Siggraph '81 Conference Proceedings.

[76] J.F. Murray-Coleman and A. M. Smith. The automated measurement of brdfs and their application to luminaire modeling. *Journal of the Illumination Engineering Society*, 19(1):87–99, January 1990.

[77] Eihachiro Nakamae, Kazufumi Kaneda, Takashi Okamoto, and Tomoyuki Nishita. A lighting model aiming at drive simulators. *Computer Graphics*, 24(3):395–404, August 1990. ACM Siggraph '90 Conference Proceedings.

[78] Laszlo Neuman and Attila Neumann. Photosimulation: Interreflection with arbitrary reflectance models and illumination. *Computer Graphics Forum*, 8:21–34, 1989.

[79] Tomoyuki Nishita and Eihachiro Nakamae. Continuous tone representation of three-dimensional objects taking account of shadows and interreflection. *Computer Graphics*, 19(3):23–30, July 1985. ACM Siggraph '85 Conference Proceedings.

[80] K. A. O'Donnell and E. R. Mendez. Experimental study of scattering from characterized random surfaces. *Journal of the Optical Society of America*, 4(7):1194–1205, July 1987.

[81] James Painter and Kenneth Sloan. Antialiased ray tracing by adaptive progressive refinement. *Computer Graphics*, 23(3):281–288, July 1989. ACM Siggraph '89 Conference Proceedings.

[82] Edward D. Palik. *Handbook of Optical Constants of Solids*. Academic Press, New York, N.Y., 1985.

[83] G. D. Parfitt and K. S. Sing, editors. *Characterization of Powder Surfaces*. Academic Press, New York, N.Y., 1976.

[84] Ken Perlin. An image synthesizer. *Computer Graphics*, 19(3):287–296, July 1985. ACM Siggraph '85 Conference Proceedings.

[85] Ken Perlin and Eric M. Hoffert. Hypertexture. *Computer Graphics*, 23(3):253–262, July 1989. ACM Siggraph '89 Conference Proceedings.

[86] Michael Potmesil and Indranil Chakravarty. A lens and aperture model for synthetic image generation. *Computer Graphics*, 15(3):297–305, August 1981. ACM Siggraph '81 Conference Proceedings.

[87] Pierre Poulin and Alain Fournier. A model for anisotropic reflection. *Computer Graphics*, 24(3):267–282, August 1990. ACM Siggraph '90 Conference Proceedings.

[88] Claude Puech, Francois Sillion, and Christophe Vedel. Improving interaction with radiosity-based lighting simulation programs. *Computer Graphics*, 24(1):51–57, 1990. ACM Workshop on Interactive Graphics Proceedings.

[89] Hazel Rossotti. *Colour: Why the World Isn't Grey*. Princeton University Press, Princeton, N. J., 1983.

[90] H. E. Rushmeier, D. R. Baum, and D. E. Hall. Accelerating the hemo-cube algorithm for calculating form factors. In *ASME Heat Transfer Conference*, pages 45–52, June 1990.

[91] Holly Rushmeier and Greg Ward. Experimental comparison and evaluation. *Radiosity*, 1990. ACM Siggraph '90 Course Notes.

[92] Holly E. Rushmeier. *Realistic Image Synthesis for Scenes with Radiatively Participating Media*. PhD thesis, Cornell University, May 1988.

[93] Holly E. Rushmeier and Kenneth E. Torrance. The zonal method for calculating light intensities in the presence of a participating medium. *Computer Graphics*, 21(4):293–302, July 1987. ACM Siggraph '87 Conference Proceedings.

[94] Holly E. Rushmeier and Kenneth E. Torrance. Extending the radiosity method to include specularly reflecting and translucent materials. *ACM Transaction on Graphics*, 9(1):1–27, January 1990.

[95] Bertrand Le Saec and Christophe Schlick. A progressive ray-tracing-based radiosity with general reflectance functions. In *Proceedings of the Eurographics Workshop on Photosimulation, Realism and Physics in Computer Graphics*, pages 103–116, June 1990.

[96] L. G. Schultz and F. R. Tangherlini. Optical constants of silver, gold, copper, and aluminum ii. the index of refraction n. *Journal of the Optical Society of America*, 44(5):362–368, May 1954.

[97] Y. A. Screider. *The Monte Carlo Method.* Pergamon Press, New York, N.Y., 1966.

[98] Peter Shirley. *Physically Based Lighting Calculations for Computer Graphics.* PhD thesis, University of Illinois at Urbana-Champaign, November 1990.

[99] Peter Shirley. Physically based lighting calculations for computer graphics: A modern perspective. In *Proceedings of the Eurographics Workshop on Photosimulation, Realism and Physics in Computer Graphics*, pages 67–81, June 1990.

[100] Peter Shirley. A ray tracing algorithm for global illumination. *Graphics Interface '90*, May 1990.

[101] Peter Shirley and Henry Neeman. Volume visualization at the center for supercomputing research and development. In *Proceedings of the Chapel Hill Workshop on Volume Visualization*, pages 17–20, May 1989.

[102] Robert Siegel and John R. Howell. *Thermal Radiation Heat Transfer.* McGraw-Hill, New York, N.Y., 1981.

[103] Francois Sillion and Claude Puech. A general two-pass method integrating specular and diffuse reflection. *Computer Graphics*, 23(3):335–344, July 1989. ACM Siggraph '89 Conference Proceedings.

[104] Brian E. Smits and Gary W. Meyer. Newton's colors: Simulating interference phenomena in realistic image synthesis. In *Proceedings of the Eurographics Workshop on Photosimulation, Realism and Physics in Computer Graphics*, pages 185–194, June 1990.

[105] Jerome Spanier and Ely M. Gelbard. *Monte Carlo Principles and Neutron Transport Problems.* Addison-Wesley, New York, N.Y., 1969.

[106] E. M. Sparrow and R. D. Cess. *Radiation Heat Transfer.* Brooks/Cole, Belmont, Ca., 1970.

[107] Stephen N. Spencer. The hemisphere radiosity method: A tale of two algorithms. In *Proceedings of the Eurographics Workshop on Photosimulation, Realism and Physics in Computer Graphics*, pages 127–135, June 1990.

[108] Dan Stanger. Monte carlo procedures in lighting design. *Journal of the Illumination Engineering Society*, pages 14–25, July 1984.

[109] A. H. Stroud. *Approximate Calculation of Multiple Integrals.* Prentics-Hall, Englewood Cliffs, N. J., 1971.

[110] Atsushi Takagi, Hiroshi Takaoka, Tetsuya Oshima, and Yoshinori Ogata. Accurate rendering technique based on colorimetric conception. *Computer Graphics*, 24(3):263–272, August 1990. ACM Siggraph '90 Conference Proceedings.

[111] Pierre Tellier and Kadi Bouatouch. Vers un modele d'eclairement realiste. Technical Report 464, IRISA, Campus de Beaulieu, April 1989.

[112] Spencer W. Thomas. Dispersive refraction in ray tracing. *Visual Computer*, 2:3–8, 1986.

[113] J. S. Toor and R. Viskanta. A numerical experiment of radiant heat interchange by the monte carlo method. *International Journal of Heat and Mass Transfer*, 11:883–897, 1968.

[114] Craig Upson and Micheal Keeler. V-buffer: Visible volume rendering. *Computer Graphics*, 22(4):59–64, July 1988. ACM Siggraph '88 Conference Proceedings.

[115] Steve Upstill. *The Renderman Companion.* Addison-Wesley, Reading, MA, 1990.

[116] H. C. van de Hulst. *Light Scattering by Small Particles.* Dover, New York, N.Y., 1981. Reprint of 1957 Wiley Edition.

[117] John R. Wallace, Michael F. Cohen, and Donald P. Greenberg. A two-pass solution to the rendering equation: a synthesis of ray tracing and radiosity methods. *Computer Graphics*, 21(4):311–320, July 1987. ACM Siggraph '87 Conference Proceedings.

[118] John R. Wallace, Kells A. Elmquist, and Eric A. Haines. A ray tracing algorithm for progressive radiosity. *Computer Graphics*, 23(3):335–344, July 1989. ACM Siggraph '89 Conference Proceedings.

[119] Gregory J. Ward, Francis M. Rubinstein, and Robert D. Clear. A ray tracing solution for diffuse interreflection. *Computer Graphics*, 22(4):85–92, August 1988. ACM Siggraph '88 Conference Proceedings.

[120] Mark Watt. Light-water interaction using backward beam tracing. *Computer Graphics*, 24(3):377–386, August 1990. ACM Siggraph '90 Conference Proceedings.

[121] Turner Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 23(6):343–349, June 1980.

[122] Samuel J. Williamson and Herman Z. Cummins. *Light and Color in Nature and Art.* Wiley, New York, N.Y., 1983.

[123] Hau Xu, Qun-Sheng Peng, and You-Dong Liang. Accelerating radiosity method for complex scenes. In *Eurographics '89*, pages 51–61, 1989.

[124] Sidney J. Yakowitz. *Computational Probability and Simulation.* Addison-Wesley, New York, N.Y., 1977.

[125] Shigeki Yokoi, Kosuke Kurashige, and Jun ichiro Toriwaki. Rendering gems with asterism or chatoyancy. *Visual Computer*, 2:307–312, February 1986.

[126] S. K. Zeremba. The mathematical basis of monte carlo and quasi-monte carlo methods. *SIAM Review*, 10(3):303–314, July 1968.