Light-Driven Global Illumination with a Wavelet Representation of Light Transport

by

Robert R. Lewis

B. S. Physics, Harvey Mudd College, 1974

M. A. Astronomy, University of California at Berkeley, 1979

M. S. Computer Science and Engineering, Oregon Graduate Institute, 1989

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

in

THE FACULTY OF GRADUATE STUDIES

(Department of Computer Science)
we accept this thesis as conforming
to the required standard

—

The University of British Columbia

February 1998

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of COMPUTER SCIENCE

The University of British Columbia
Vancouver, Canada

Date 6/17/98

Abstract

This thesis considers the problem of global illumination: the modelling of light as it travels through a scene interacting with the objects contained within the scene. Starting with a description of the problem and a discussion of previous work, we explore a new approach called *light-driven global illumination* that offers several advantages over its predecessors: a lower asymptotic complexity, a wider range of representable surface interaction phenomena, and an absence of the need for "meshing" – object surface subdivision needed primarily to represent shadows.

Light-driven global illumination is intermediate between local and global illumination. Representing light with wavelet basis functions, we are able to treat both the interaction between two surfaces and the interaction of a surface with a radiation field in a source-to-destination model that applies to whole surfaces, not just small elements.

We have found this "wavelet radiative transfer" to be a valid way to generate and store complex global light field data as four-dimensional textures for incorporation in local illumination solutions. Wavelets can considerably reduce the otherwise substantial storage and reconstruction problems associated with doing this. We include several examples of this.

We also discuss plausible illumination models, which are required to make

light-driven global illumination work theoretically. Like wavelet radiative transfer, these models have application in other areas of rendering besides global illumination.

Finally, we develop the theory behind light-driven global illumination and apply it successfully to some simple examples. While we find the algorithm to be quite slow compared to other well-known rendering algorithms, we analyze what is needed to make it competitive.

In conclusion, we find that representing light with wavelets has a set of advantages that are independent of the comparative inefficiency of the light-driven algorithm.

Contents

List of Tables

List of Figures

Acknowledgements

The author is grateful to John Buchanan, Pat Hanrahan, Paul Lalonde, Pierre Poulin, Leena-Maija Reissell, and, of course, his supervisor, Alain Fournier, for encouragment and numerous discussions on this thesis and related subjects.

ROBERT R. LEWIS

The University of British Columbia

February 1998

For Victoria,

for the patience, the long hours, and, of course, the love.

And for Alain,

EDGAR BERGEN: Mortimer, I can't understand

how one person can possibly be so stupid!

MORTIMER SNERD: Well, yuh see Mr. Bergen,

I got this fella helpin' me...

Chapter 1

Introduction

Realistic images are images which are indistinguishable from images captured from the physical world. The creation of such images is a fundamental goal in computer graphics. Research in the modelling of shape, of properties of surfaces, and of the interaction of light with matter is directed towards this goal, as evidenced in major texts on the subject such as Glassner [24], Foley, et al. [18], and Watt [80].

For the past decade, some of the most realistic images in computer graphics have been produced by global illumination techniques. These techniques are limited, however, in the range of lighting and surface characteristics that they can represent.

This thesis presents a new technique in realistic image synthesis called "light-driven global illumination" that allows a much wider range of characteristics to be modelled. After this introduction, we will examine the basics of local and global illumination in Chapter 2. Chapter 3 will present the new global illumination algorithm itself. Chapter 4 will present an analysis of the "plausibility" of current illumination models. (As we will see, energy-conserving illumination models, which are a subset of physically plausible illumination models, are a theoretical prereq-

uisite for our technique.) Chapter 5 explains the approach we use to perform the local solutions required by the global solution, a formulation of radiative transfer and surface interaction that is based on wavelet techniques. Chapters 6 and 7 describes implementations of ideas from the preceding chapters. Chapter 8 describes our conclusions.

1.1 Conventions

Table 1.1 lists symbols that will be commonly used throughout this thesis. Equations taken from other sources will be modified to use these symbols.

Table 1.1: Table of Commonly-Used Symbols. The "*" is a wildcard matching character.

symbol	definition
α	angle between $\mathbf{N}$ and $\mathbf{H}$
$\mathcal{B}$	blocked radiative transport operator
b_*	scaling constants for various illumination models
β	specular "half-angle"
dA	element of surface area
$d\omega_i$	$\sin\theta_i d\theta_i d\phi_i$, an element of incident solid angle
$d\omega_r$	$\sin\theta_r d\theta_r d\phi_r$, an element of reflected solid angle
D_*	facet slope distribution functions
E	irradiance
E_N	normal irradiance
F	Fresnel factor
f_r	bidirectional reflectance distribution function (BRDF)
F_s^*	specular shading functions
ϕ_i	incident azimuthal angle
ϕ_r	reflected azimuthal angle
Φ_L	angle between incident and viewing directions
$\mathcal{G}$	global illumination operator
G	geometrical attenuation factor
$\mathbf{H}$	bisector of source and viewing directions
$\mathcal{I}$	identity operator

continued on next page

Table 1.1: Table of Commonly-Used Symbols. The "*" is a wildcard matching character.

continued from previous page	
symbol	definition
k_a	ambient reflectance coefficient
k_d	diffuse reflectance coefficient
k_s	specular reflectance coefficient
k_ρ^*	directional-hemispherical reflectances
k_σ	fraction of total energy reflected specularly
L_a	ambient radiance
L_i	incident radiance
L_r	reflected radiance
L_t	transmitted radiance
M	exitance
m	RMS slope of the surface
$\mathbf{N}$	the surface normal
n_s^*	specular exponents
$\Omega_{\mathbf{N}}^+$	the hemisphere surrounding $\mathbf{N}$
$\Omega_{\mathbf{N}}^-$	the hemisphere surrounding $-\mathbf{N}$
$\mathbf{R}_l$	reflected incident direction
$\mathcal{S}$	surface interaction operator
$\mathbf{S}$	incident direction
$\mathcal{T}$	radiative transport operator
	continued on next page

Table 1.1: Table of Commonly-Used Symbols. The "*" is a wildcard matching character.

continued from previous page	
symbol	definition
θ_i	incident polar angle
θ_r	reflected polar angle
θ_t	transmitted polar angle
$\mathbf{V}$	direction of viewer

Physics	Radiometry	Radiometric Units (SI)
	radiant energy	joules [$J = kgm^2s^{-2}$]
flux	radiant power	watts [$W = Js^{-1}$]
angular flux density	radiance	[$Wm^{-2}sr^{-1}$]
incident flux density	irradiance	[Wm^{-2}]
emergent flux density	radiosity (or exitance)	[Wm^{-2}]
	radiant intensity	[Wsr^{-2}]

Table 1.2: Light Measurement Terminology

We have attempted to be compatible with the ANSI/IES standard [36] wherever possible. Table 1.2, taken from Cohen and Wallace [14] with minor alterations, lists the equivalent terminology in the physics and radiometry (the science of the physical measurement of electromagnetic energy) communities of the quantities we will be discussing.

We will not be considering photometry (the psychophysical measurement of the visual sensation produced by the electromagnetic spectrum), as determining what viewers actually *see* when looking at a scene. This may be treated as an orthogonal postprocessing step after an accurate computation of the light that reaches them. The latter is our goal.

Chapter 2

Illumination

This chapter will describe the theory of illumination that underlies rendering and then goes on to an overview of previous work in global illumination in computer graphics.

2.1 What is Illumination?

The study of illumination is the study of light and how it interacts with matter to produce visible scenes. Its principles are derived from those of physics. As we will see in subsequent chapters, the equations describing illumination are relatively easy to write down, but hard to solve in non-trivial cases.

2.2 Relation to Computer Graphics

Computer graphics is the use of computers to produce images. Frequently, it is desirable that those images look realistic. "Realistic" in our context means that viewers get the impression that what they are seeing when they look at a computer-

generated (i.e., *rendered*) artifact (photograph, video screen, etc.) is an image of a scene that exists in the physical world outside the computer.

It does not mean that they receive the same stimulus that they would in a physical setting. After all, it is usually quite obvious when one is looking at a photograph or a video image. What we want is for the artifact to be indistinguishable from a photograph or video image of a scene in the physical world: to be *realistic.* It is therefore productive to study illumination for this purpose.

2.3 Local and Global Illumination

There are two broad categories of problems that arise when attempting to solve for illumination: local and global.

Local illumination problems are confined to small (often infinitesimal) areas or volumes. They usually involve a single light source (which we will refer to as a *luminaire*), a surface or volume element, and a viewing direction. The basic question of local illumination is: How does the element interact with the incident light to produce the light leaving the element in the viewing direction?

Global illumination describes how light is distributed in a scene: a collection of objects, including luminaires, immersed in a given medium. Figure 2.1 shows a typical global illumination problem. Global illumination solutions must consider multiple reflections. Part of that solution may invoke local illumination solutions. Fournier [22] describes their interrelation further.

The goal of a global illumination scheme may vary with the needs of the user. In order of increasing scope, global illumination problems may require solution at...

- *a small set of points.* For example, the user may only need to know if there

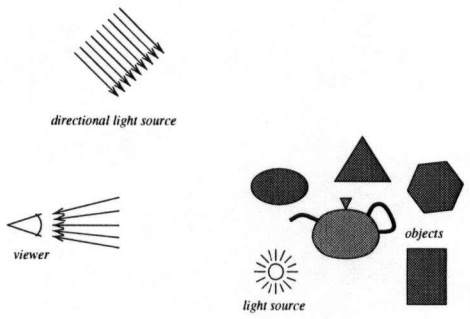

Figure 2.1: A Typical Global Illumination Problem

is sufficient light for people seated in a library to read by. (These sorts of problems are more often found in illumination engineering than in computer graphics.)

- *all visible surfaces.* Given a scene with non-participating media and a viewer at a fixed location, compute the illumination of every surface that the viewer sees. (This is the "viewer dependent" problem shown in Figure 2.1.)

- *all surfaces.* Like the previous class, but producing a solution that represents the illumination on every surface, which, for example, allows the user to do a "walkthrough" as a postprocessing step. (This is the "viewer independent" problem.)

- *all points.* In the presence of participating media, it is in general necessary to compute illumination not just at surfaces, but throughout the volume of a scene.

2.4 Radiative Transport Theory

This section will present the physical basis underlying global illumination as modelled by computer graphics. It will start with a discussion of radiance and how it relates to other physical quantities,

2.4.1 Radiance and the Transport Operator $\mathcal{T}$

Let us first discuss some of the basics of how light is represented. The fundamental quantity is *radiance*, which is defined in the ANSI/IES standard [36] to be the amount of power passing in a given direction though a given surface per unit area (perpendicular to the direction of travel) per unit solid angle. Radiance at a point $\mathbf{P}$ in a direction $\mathbf{S}$ is usually represented by a function $L(\mathbf{P}, \mathbf{S})$.

We note, as the standard does, a distinction between "radiance" and "spectral radiance". Spectral radiance adds additional units of inverse wavelength to radiance to allow for a distribution of radiance over wavelength.

This thesis (like most of the rendering literature) takes radiance to be monochromatic and assumes one can construct a polychromatic image by combining "channels" of independently computed monochromatic images. We assume further that these channels are "independent": that they do not interact with each other, as would be the case in the presence of phenomena like fluorescence. Also like most of the literature, we ignore polarization and assume time invariance.

One very useful property of radiance, or "scene radiance" as it is called, is its proportionality to "image irradiance", the quantity detected by a camera viewing a scene. This relationship is derived in Chapter 10 of Horn [34].

Radiance's most useful property for rendering, however, is its invariance: In a non-participating medium and in the absence of diffraction phenomena (both

of which we assume here), the radiance given off at a point $\mathbf{P}_o$ on a surface in a direction $\mathbf{S}$ is constant until reaching another surface[1].

We can express this as the action of a *radiative*[2] *transport operator* $\mathcal{T}$ acting on a source radiance L_s to produce a destination radiance L_d:

$$L_d(\mathbf{P}, \mathbf{S}) = \mathcal{T}L_s = L_s(\mathbf{P}_o(\mathbf{P}, \mathbf{S}), \mathbf{S}) \tag{2.1}$$

Since we also assume that the radiation travels in a straight line[3],

$$\mathbf{P} = \mathbf{P}_o + \mathbf{S}t \tag{2.2}$$

for some scalar $t > 0$. This principle underlies raytracing.

Technically, we should allow for the light travel time from source to destination, but our assumption of time invariance makes this unnecessary. We can therefore treat light travel as instantaneous.

It is evident from Equation (2.1) that for any scaling factor α and radiance distribution L,

$$\mathcal{T}\alpha L = \alpha \mathcal{T} L. \tag{2.3}$$

$\mathcal{T}$ is also linear in the following sense: given two sources A and B with radiances L_{sA} and L_{sB} and $L_{dA} = \mathcal{T}L_{sA}$ and $L_{dB} = \mathcal{T}L_{sB}$, then the resulting radiance L_d is

$$L_d(\mathbf{P}, \mathbf{S}) = \mathcal{T}\left(L_{sA} + L_{sB}\right) = \mathcal{T}L_{sA} + \mathcal{T}L_{sB} = L_{dA} + L_{dB} \tag{2.4}$$

if, from the point of view of $\mathbf{P}$, neither A nor B obstructs the other.

[1] This is derived, for instance, in Siegel and Howell [68].

[2] For brevity, we may drop the "radiative" throughout this thesis, since we are only concerned with radiative transport, as opposed to conductive, convective, or any other form of energy transport.

[3] Note that we are not ruling out refraction, which takes place at a surface, not during propagation.

Let us review the assumptions we have made about radiance and $\mathcal{T}$. These are all based on the way light behaves under what we consider "ordinary" rendering circumstances:

- straight-line (ray) propagation
- linearity (in the absence of blocking)
- instantaneous transport
- energy conservation
- steady-state power transfer
- non-interacting monochromatic channels
- absence of diffraction phenomena
- absence of polarization phenomena

It is important to note that while all of these assumptions are compatible with classical electromagnetic theory (under certain circumstances), it is these empirically-based assumptions that drive our discussion, not the theory itself. Other areas of investigation which make similar assumptions (e.g., neutron transport) may find this work applicable.

2.4.2 The Blocking Operator $\mathcal{B}$

We noted in Section 2.4.1 the complications that can arise in the presence of more than one destination object. We can account for them by the incorporation of a *blocking operator* $\mathcal{B}$ defined for an obstructing object O:

$$L_b(\mathbf{P},\mathbf{S}) = \mathcal{B}L_s = b(\mathbf{P},\mathbf{S})L_s(\mathbf{P},\mathbf{S}) \tag{2.5}$$

where $b(\mathbf{P}, \mathbf{S})$ is a geometric function which is 0 if a ray starting on the source object at point $\mathbf{P}$ in direction $\mathbf{S}$ intersects O and 1 otherwise.

If we then have a source object S and a destination object D, and if O does not lie "behind" D, we can express the resulting radiance L_d as

$$L_d = \mathcal{T}\mathcal{B}L_s. \tag{2.6}$$

2.4.3 Reflection, Transmission, and the Surface Interaction Operator $\mathcal{S}$

In this section, we will review the formulation of the fundamental equation describing physically-based illumination modelling, presented below as Equation (2.11). Our presentation follows those given in Cook and Torrance [15], Foley, et al. [18], and Cohen and Wallace [14].

Figure 2.2 shows the (usual) geometry for two cases of local illumination. In both of them, an infinitesimal element of surface area dA is being illuminated by an incident radiance L_i coming from an infinitesimal solid angle $d\omega_i$ surrounding direction $\mathbf{S}$. (All vectors presented here are are of unit magnitude.) An observer (or measuring device) is located in direction $\mathbf{V}$. $\mathbf{N}$ is the surface normal at dA.

On the left, the observer is above the "horizon" of dA in the hemisphere $\Omega^+_{\mathbf{N}}$, the hemisphere surrounding $\mathbf{N}$. In this case, the observer receives radiation reflected from dA. On the right, the observer is below the horizon in the hemisphere $\Omega^-_{\mathbf{N}}$ surrounding $-\mathbf{N}$. In this case, the observer receives radiation transmitted by dA.

As treatment of these two cases is very similar, let us consider the reflection case first. We assume all reflected radiation passes unobstructed into $\Omega^+_{\mathbf{N}}$.

We use $\mathbf{N}$ as the z-axis of a polar coordinate system so that we can specify

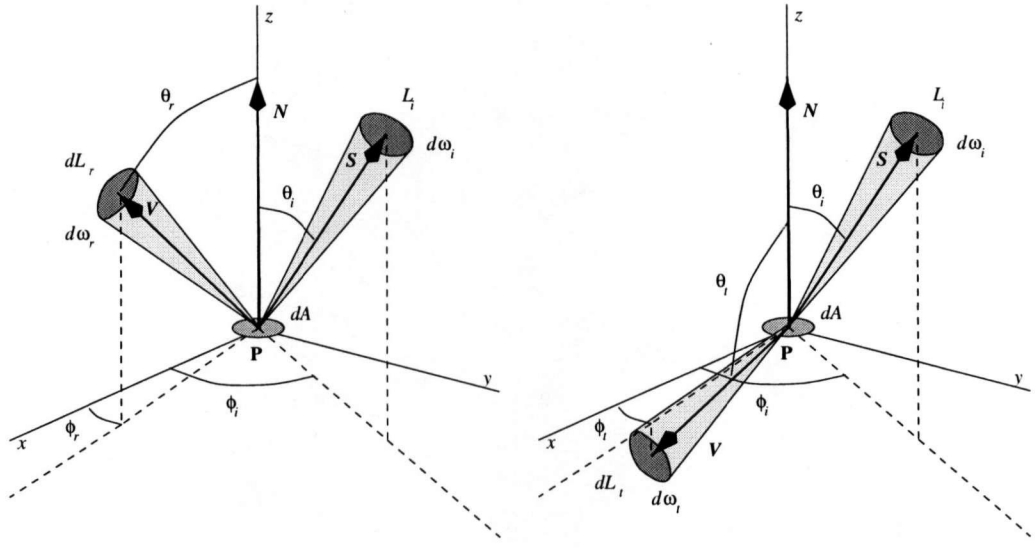

Figure 2.2: Surface Geometry for Reflection and Transmission

S by the usual incident polar and azimuthal angles θ_i and ϕ_i and **V** by the usual reflected polar and azimuthal angles θ_r and ϕ_r[4].

To consider an illumination model from the standpoint of power balance, we start from the equation of power balance given in Siegel and Howell [68]:

$$dE = L_i \, (\mathbf{N} \cdot \mathbf{S}) \, d\omega_i \tag{2.7}$$

where dE is the irradiance resulting from the illumination of the patch in direction **S** subtending the solid angle $d\omega_i$. Using the definition of f_r, the bidirectional reflectance distribution function (hereafter, BRDF), this irradiance gives rise to dL_r, the resulting radiance of the patch:

$$dL_r = f_r(\mathbf{S}, \mathbf{V}) \, dE \tag{2.8}$$

[4]For the time being, we will assume ϕ_i and ϕ_r are measured from some locally-defined frame of reference.

In general, f_r is a function of the incident direction (θ_i, ϕ_i) and the reflected direction (θ_r, ϕ_r)[5]. For the sake of brevity, we will write it as $f_r(\mathbf{S}, \mathbf{V})$.

As we have assumed an opaque, nonemissive surface, the only contributions to L_r can come from $\Omega_{\mathbf{N}}^+$. Therefore, we can integrate $d\omega_i$ over $\Omega_{\mathbf{N}}^+$ to get

$$L_r = \int_{\Omega_{\mathbf{N}}^+} f_r(\mathbf{S}, \mathbf{V})\, L_i\, (\mathbf{N} \cdot \mathbf{S})\, d\omega_i \tag{2.9}$$

We allow for transmission, as shown on the right of Figure 2.2, by considering that radiation can come from anywhere and can go in any direction. This means that we need to generalize the limits of our directional integration to refer not to $\Omega_{\mathbf{N}}^+$ and $\Omega_{\mathbf{N}}^-$, but to $\Omega_{\mathbf{N}}^R$, the hemisphere (either $\Omega_{\mathbf{N}}^+$ or $\Omega_{\mathbf{N}}^-$) that contains $\mathbf{V}$ and $\Omega_{\mathbf{N}}^T$, the hemisphere opposite $\Omega_{\mathbf{N}}^R$. We also need to allow for a transmitted component of radiance using the BTDF f_t. Equation (2.9) then becomes

$$L_t = \int_{\Omega_{\mathbf{N}}^T} f_t(\mathbf{S}, \mathbf{V})\, L_i\, |\mathbf{N} \cdot \mathbf{S}|\, d\omega_i \tag{2.10}$$

In general, then, the total radiance for a non-emissive surface is $L = L_r + L_t$. If the surface is emissive, we can add an additional surface emissivity term L_e when the observer is in $\Omega_{\mathbf{N}}^+$[6]. To summarize, then, the total radiance given off by dA is

$$\begin{aligned} L_r(\mathbf{V}) &= L_e(\mathbf{V}) + \int_{\Omega_{\mathbf{N}}^R} f_r(\mathbf{S}^+, \mathbf{V})\, L_i(\mathbf{S}^+)\, |\mathbf{N} \cdot \mathbf{S}^+|\, d\omega_i \\ &+ \int_{\Omega_{\mathbf{N}}^T} f_t(\mathbf{S}^-, \mathbf{V})\, L_i(\mathbf{S}^-)\, |\mathbf{N} \cdot \mathbf{S}^-|\, d\omega_i \end{aligned} \tag{2.11}$$

where L_r is the total radiance given off of a surface with normal $\mathbf{N}$, L_e is the surface emissivity, L_i is the incident radiance, $\Omega_{\mathbf{N}}^R$ is the reflection hemisphere (contains

[5] f_r may also vary over the surface, but that variation is usually treated as part of texturing rather than illumination modelling, so we will ignore it here. As mentioned above, we are also ignoring its possible dependence on wavelength.

[6] $\Omega_{\mathbf{N}}^-$ is somewhat trickier, since if the surface is emissive, then the interior of the object is also likely to be emissive, which would imply a volume emissivity rather than a surface emissivity. As we are ignoring participating media in this section, we will therefore restrict ourselves to surface emissivity only.

$\mathbf{N}$, the "viewing" direction $\mathbf{V}$, and the "positive source" direction $\mathbf{S}^+$), $\Omega_{\mathbf{N}}^T$ is the transmission hemisphere (opposite $\Omega_{\mathbf{N}}^R$, containing the "negative source" direction $\mathbf{S}^-$), f_r is the BRDF, f_t is the bidirectional transmittance distribution function (BTDF), $d\omega_i$ is $\sin\theta_i d\theta_i d\phi_i$, θ_i is the incident polar angle, and ϕ_i is the incident azimuthal angle.

A local illumination solution is entirely characterized by its BRDF and BTDF functions. Note that we ignore the spatial variation of the radiances and distribution functions, since it symbolically factors out of Equation (2.11),

We represent combined reflection and transmission surface interaction by the integral operator $\mathcal{S}$:

$$L_r = L_e + \mathcal{S}L_i \tag{2.12}$$

2.5 Renderers

In computer graphics, a realistic rendering package (which we will refer to hereafter as a *renderer*) is a (predominantly) software system used to create realistic images. It usually does this by performing a simplified solution of the equations describing light and matter interaction.

It is useful to consider a rendering package as being composed of seven parts:

- a *luminaire model*: how light sources are represented
- a *propagation model*: how light travels from source to surface, surface to surface, and surface to viewer
- an *object model*: how objects in the scene are represented
- an *illumination model*: how light interacts with the surface of an object

- a *surface model*: how small-scale surface structure is represented
- a *shading model*: how light passing to the viewer is converted into an image
- a *rendering technique*: how all the models that constitute the package interact to produce an image

All of these are necessary for a rendering package, although in some cases they can be trivial.

In the remainder of this section, we will discuss each of these parts and how they relate to each other.

2.5.1 Luminaire Models

Models of luminaires, or light sources, can range from the simple to the sophisticated. In this section, we will cover several popular ones. The most important attribute of a luminaire model is its dimensionality: 0 (a point luminaire), 1 (a linear luminaire), or 2 (an area luminaire).

2.5.1.1 Point Luminaire

A point luminaire assumes that a finite amount of light is coming from a single point. If we take Φ as the amount of emissive power being given off by the luminaire, the radiance of a point luminaire is:

$$L_e = \frac{\Phi}{4\pi r^2}\delta(\mathbf{S} - \mathbf{S}') \tag{2.13}$$

where r is the distance from the luminaire to the illuminated point and S' is the direction from the illuminated point to the luminaire. The Dirac delta function $\delta(x)$ is defined by

$$\int_{-\infty}^{\infty} f(x)\delta(x)dx = f(0) \tag{2.14}$$

for any function $f(x)$.

2.5.1.2 Directional Luminaire

A directional luminaire is effectively a point luminaire at infinity: every point on the illuminated surface receives light from the same direction. Unlike a point luminaire, however, a directional luminaire is not characterized by its power (which is infinite) and a position, but by its *normal irradiance* E_N, the amount of power per unit area transferred to a surface with normal incidence[7], and a fixed direction $\mathbf{S}'$. We can express the radiance of a directional luminaire as

$$L_e = E_N \delta(\mathbf{S} - \mathbf{S}'). \tag{2.15}$$

2.5.1.3 Other Luminaires

Directional and point luminaires were the only ones available in early renderers (with the exception of diffuse area luminaires used in radiosity). Verbeck and Greenberg, for example, describe in [77] how to construct approximation of linear and area light sources from point light sources. They also describe how to model the effects of flaps and focussing.

In 1990, Poulin and Amanatides [59] were able to model linear luminaires directly. The next year, Tanaka and Takahashi [74] did the same thing for area luminaires. Both of these groups worked with sources that were isotropic in the sense that the emitted radiance was constant with respect to direction for each differential line or area element.

In 1992, Ashdown [2] drew on work from the illumination engineering community to incorporate measurements of real, non-isotropic luminaires using "near-

[7] E_N is the value of irradiance E when the light is coming from the normal direction N.

field photometry". More recently, Lalonde and Fournier [41] was able to efficiently represent an area luminaire using wavelets.

2.5.1.4 What Constitutes a Luminaire?

The distinction between a luminaire and any other object is somewhat artificial. Light interacting with a surface produces reflected and transmitted radiance distributions which, as far as a renderer is concerned, should be indistinguishable from emitted distributions. The technique we will present in this thesis will make no (essential) distinctions between emitted, reflected, and transmitted radiance distributions.

2.5.2 Propagation Models

Light travels from surface to surface through a medium. The easiest medium to represent is a non-participating medium – a medium that does not in any way impede radiation travelling through it. A vacuum is one such medium.

The propagation model in such a medium is the one we used to describe the invariance of radiance under the $\mathcal{T}$ operator in Equation (2.1): the radiance remains constant along a ray from source to destination.

In the presence of a participating medium, propagation becomes more difficult to represent. There are three phenomena which must be taken into account: emission, absorption, and scattering.

We have already mentioned emission from surfaces, but it can also happen from an incandescent gas such as a fire, a gaseous nebula, or the Sun. When the model of emission cannot be confined to a two-dimensional manifold, it must be treated as part of propagation.

Absorption removes energy from the beam entirely, converting it to a non-visible form of energy such as heat. Smoke, for instance, can make a very absorptive medium.

Scattering does not eliminate energy from the beam, but redirects it. It may remove energy along the line-of-sight from source to destination (outscattering), but it may also scatter energy into the line-of-sight from other sources (inscattering). So, for instance, a street light in fog, a typical scattering medium, appears surrounded by a halo of light that was redirected towards the viewer by the medium.

Allowing for participating media turns Equation (2.1) into an integro-differential equation. Taking Glassner's [24] formulation of it and substituting our own notation, we get:

$$\begin{aligned} \mathbf{S} \cdot \nabla L(\mathbf{P}, \mathbf{S}) &= j(\mathbf{P}, \mathbf{S}) \\ &+ \int_{\Omega} \kappa(\mathbf{P}, \mathbf{S}' \rightarrow \mathbf{S}) L(\mathbf{P}, \mathbf{S}') d\omega' \\ &- L(\mathbf{P}, \mathbf{S}) \left[\int_{\Omega} \kappa(\mathbf{P}, \mathbf{S} \rightarrow \mathbf{S}') d\omega' + \sigma_a(\mathbf{P}, \mathbf{S}) \right] \end{aligned} \tag{2.16}$$

where j is the volume emissivity, Ω is the directional sphere, κ is the volume outscattering or inscattering probability function, ω_a is absorption function.

Equation (2.16) is easy to interpret. The left hand side is the spatial variation of radiance in direction $\mathbf{S}$. On the right hand side, the first term is the volume emissivity adding power (radiance per unit length, actually) to the beam. The second is the amount of power scattered into the beam. The third is the sum of the power scattered out of the beam and the amount of power absorbed by the medium.

A considerable amount of work has been done in the fields of physics (e.g. Chandrasekhar [10]), astronomy (e.g. Mihalas [51]), oceanography (e.g. Mobley [53]), and elsewhere to understand participating media. Little, however, has been

done in computer graphics. See Rushmeier [63] for an overview of these.

2.5.3 Object Models

The construction of object models is what is normally referred to as "modelling" in computer graphics parlance. The principle role of object models is to define *surfaces*: places where radiance distributions change dramatically over an interval of path length so small that it may be considered infinitesimal. The surface of a properly-defined object subdivides space into an "inside" and an "outside".

2.5.3.1 Polygons

Polygons, especially triangles, are by far the most common object modelling construct. Each N-sided polygon is specified by a list of N vertices. Three-dimensional objects may be modeled with polyhedra, i.e., sets of polygons. Each polygon represents a "facet" of the object. By convention, the vertex list is specified in counterclockwise order when the facet is viewed from "above".

Special care must be taken to ensure that the object that results from the union of all the facets has a consistent inside and an outside, Otherwise, most renderers produce a confusing result.

2.5.3.2 Parametric Surfaces

Parametric surfaces are made up of "patches": sets of points $\{P\}$ defined by an equation of the form

$$\mathbf{P} = \mathbf{P}(u, v) \tag{2.17}$$

where $0 \leq u \leq 1$ and $0 \leq v \leq 1$.

Although there have been attempts made to render patches directly, such as Shantz and Lien [67] and Lane et al. [43], it typically proves efficient to convert each patch to a set of polygons.

One of the most widely-used algorithms for this is due to Lane et al. [43]. Given a patch, the algorithm tests it for planarity within a specified tolerance. If it is within the tolerance, the patch is replaced by a corresponding polygon. If the tolerance is exceeded, the patch is subdivided using well-known subdivision procedures and the algorithm is recursively applied.

2.5.3.3 Implicit Surfaces

An implicit surface is made up of the set of all points $\{P\}$ which satisfy

$$f(\mathbf{P}) = 0 \tag{2.18}$$

for a function f. Like parametric surfaces, implicit surfaces are generally converted to polygons for the purposes of rendering. One of the most widely-used approaches is the "marching cubes" algorithm, developed independently by Wyvill, et al. in 1986 [86] and by Lorensen and Cline in 1987 [48]. This algorithm voxellizes object space and then uses the value of f at each of the eight vertices of each voxel to generate polygons that span the voxel.

One of the shortcomings of marching cubes and similar algorithms is a sensitivity to sampling problems: Structures with spatial frequencies above the Nyquist limit may not be polygonalized correctly. In particular, there is no guarantee of preserving the topology of the polygonalized object. Recent work, however, by Stander and Hart [73] shows promise of alleviating this problem.

2.5.3.4 Constructive Solid Geometry

A notable extension of implicit surfaces is "constructive solid geometry" or CSG. It extends the definition given in Equation (2.18) to use f to classify any point in Euclidean 3-space as being inside ($f < 0$), on ($f = 0$) or outside ($f > 0$) an object. It is thereby possible to define composite objects in terms of set operations acting on such primitives. A comprehensive treatment of CSG is the book by Mäntylä [49].

CSG represents a convenient way to model objects in some but not all application domains. In addition, CSG objects are easily rendered with a raytracer (discussed in Section 2.5.7.3). Faster rendering can be achieved either by applying the marching cubes algorithm to polygonalize CSG objects or by performing CSG on polygonalized primitive objects, but just as with implicit surfaces, sampling problems can occur.

2.5.4 Illumination Models

We will review several illumination models in use in computer graphics. Space does not permit us to consider some of the more elaborate ones (such as He, et al. [33] or Oren and Nayar [57]), but as these are not yet in wide use, perhaps the reader will forgive the omission.

Our discussion will center on reflection models, as most renderers in computer graphics pay little attention to transmission.

2.5.4.1 Phong Illumination Models

Phong illumination models are generally of the form (see, for example, Foley, et al. [18], equation [16.15])

$$L_r = k_a L_a + [k_d (\mathbf{N} \cdot \mathbf{S}) + k_s F_s(\mathbf{S}, \mathbf{V})] E_N \tag{2.19}$$

where k_a is the "ambient" reflection coefficient, L_a is an ambient radiance uniformly distributed over $\Omega^+_{\mathbf{N}}$[8], k_d is the diffuse reflection coefficient, k_s is the specular reflection coefficient, and E_N is the normal irradiance parameterizing a directional luminaire.

There are two popular choices for F_s. The original one given in Phong [58] corresponds to

$$F_s^P(\mathbf{S},\mathbf{V}) = \begin{cases} (\mathbf{R}_l \cdot \mathbf{V})^{n_s^P} & \text{if } (\mathbf{R}_l \cdot \mathbf{V}) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.20}$$

where $\mathbf{R}_l = 2\mathbf{N}(\mathbf{N}\cdot\mathbf{S}) - \mathbf{S}$ is the "reflected" source direction and n_s^P is the Phong specular reflection exponent.

The other popular choice for F_s comes from Blinn [6]:

$$F_s^B(\mathbf{S},\mathbf{V}) = (\mathbf{N}\cdot\mathbf{H})^{n_s^B} \tag{2.21}$$

where $\mathbf{H} = (\mathbf{S}+\mathbf{V})/|\mathbf{S}+\mathbf{V}|$ is the unit vector halfway between $\mathbf{S}$ and $\mathbf{V}$ and n_s^B is the Blinn specular reflection exponent.

Since we are not considering emissivity or transmission in this section, let us see how these illumination models correspond to Equation (2.9).

Let the patch be illuminated by the combination of a directional source of normal irradiance E_N in direction $\mathbf{S}$ (i.e., $\theta_i = \theta_{\mathbf{S}}$, $\phi_i = \phi_{\mathbf{S}}$) on $\Omega^+_{\mathbf{N}}$ and a radiance L_a constant over $\Omega^+_{\mathbf{N}}$. Using Equation (2.15), we can model the resulting incident radiance as

$$L_i = L_a + E_N\,\delta(\cos\theta_i - \cos\theta_{\mathbf{S}})\,\delta(\phi_i - \phi_{\mathbf{S}}). \tag{2.22}$$

If we substitute this into Equation (2.9), we get

$$L_r = L_a \int_{\Omega^+_{\mathbf{N}}} f_r(\mathbf{S},\mathbf{V})\,(\mathbf{N}\cdot\mathbf{S})\,d\omega_i + E_N\,f_r(\mathbf{S},\mathbf{V})\,(\mathbf{N}\cdot\mathbf{S}) \tag{2.23}$$

[8] For the sake of simplicity, we have assumed that dA has an unobstructed view of $\Omega^+_{\mathbf{N}}$.

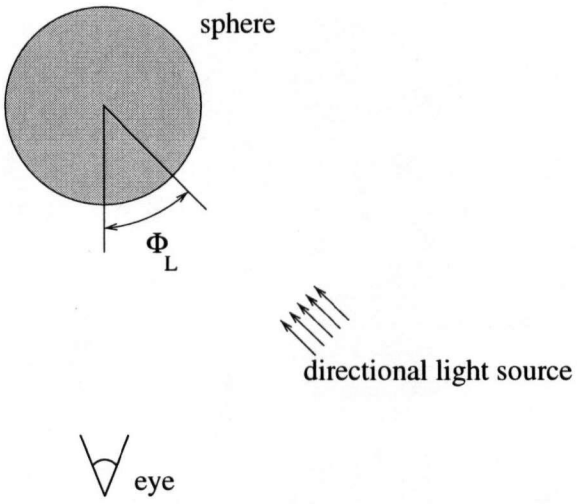

Figure 2.3: Illumination Model Test Configuration

Our goal is to make Equations (2.19) and (2.23) equivalent for all possible L_a and E_N. This means that the non-ambient terms of both equations must be equal, so

$$f_r(\mathbf{S}, \mathbf{V}) = k_d + k_s \frac{F_s(\mathbf{S}, \mathbf{V})}{(\mathbf{N} \cdot \mathbf{S})} \tag{2.24}$$

This will allow us to convert a Phong illumination model into its corresponding BRDF.

For the purposes of comparison of the various models, let us adopt the (common) test configuration whose geometry is shown in Figure 2.3. A single directional luminaire shines at a sphere. Viewed from the center of the sphere, the luminaire is located at angle Φ_L from the viewing direction.

Figure 2.4 shows a series of images generated with the test configuration with a Phong illumination model using F_s^B. In this series, k_s varies between 0 and 1 and k_d is taken to be $1 - k_s$. The angular distribution of the specular peak is

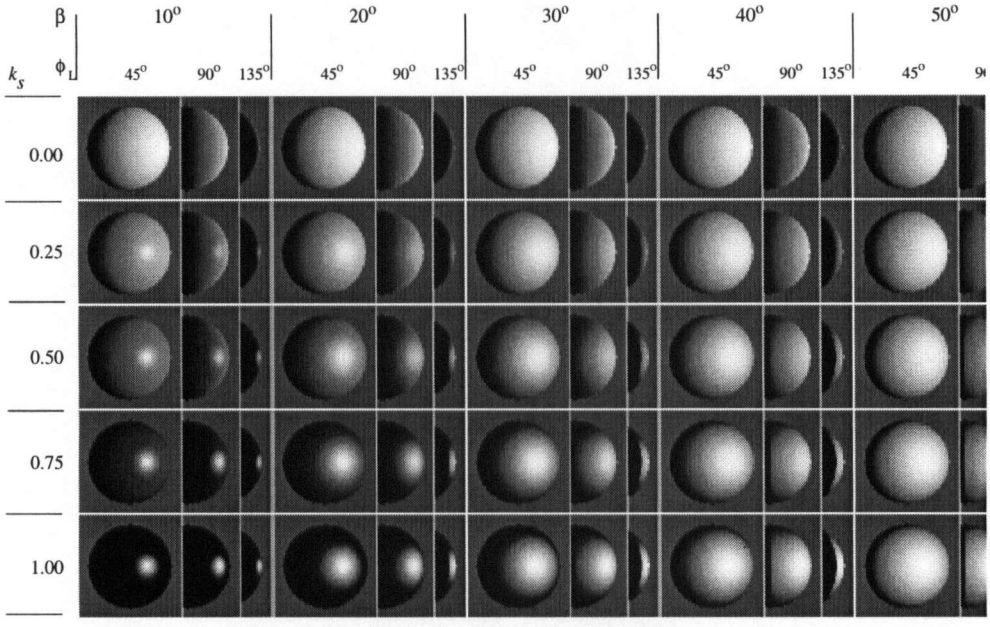

Figure 2.4: Sphere shaded with a Phong illumination model, using Blinn's F_s^B, for an assortment of incident angles with respect to the viewer (Φ_L), specular coefficients (k_s), and specular distribution half-angles (β).

qualitatively characterized[9] by the specular half-angle β, defined by

$$F_s^B = \frac{1}{2} = \cos^{n_s^B} \beta \tag{2.25}$$

Hence, for a given β,

$$n_s^B = -\frac{\ln 2}{\ln \cos \beta} \tag{2.26}$$

The ambient terms of Equations (2.19) and (2.23) must also be equal[10]. If we equate these and notice that for $\gamma \neq -1$

$$\int_{\Omega_{\mathbf{N}}^+} (\mathbf{N} \cdot \mathbf{V})^\gamma \, d\omega_r = \frac{2\pi}{\gamma + 1} \tag{2.27}$$

[9] Only in the case of Phong illumination modelling with F_s^P does β have a direct and obvious geometrical interpretation, but, as we will see (and Blinn [6] points out), it is qualitatively useful in other cases.

[10] We can look at this as a consistency constraint: The same BRDF we use to shade a directional luminaire must also shade an ambient light source.

we get

$$k_a = k_d \pi + k_s \int_{\Omega_{\mathbf{N}}^+} F_s(\mathbf{S}, \mathbf{V}) d\omega_i \tag{2.28}$$

So consistency demands that we have only two degrees of freedom in selecting k_a, k_d, and k_s. In what follows, we will take k_a to be dependent upon k_d and k_s.

2.5.4.2 Torrance-Sparrow Illumination Models

The other major class of illumination models was first proposed in Torrance and Sparrow [75] and applied to computer-generated imagery in Blinn [6]. We shall, however, follow the development given in Cook [15].

Torrance-Sparrow illumination models can be formulated directly in terms of their BRDF:

$$f_r(\mathbf{S}, \mathbf{V}) = \frac{FGD}{\pi\,(\mathbf{N} \cdot \mathbf{S})\,(\mathbf{N} \cdot \mathbf{V})} \tag{2.29}$$

where F is the Fresnel coefficient, G is the geometrical attenuation factor, and D is the facet slope distribution function.

The Fresnel coefficient for unpolarized light and zero extinction ([15] ignores extinction) is

$$F = \frac{(g-c)^2}{2(g+c)^2}\left[1 + \frac{(c\,(g+c)-1)^2}{(c\,(g-c)+1)^2}\right] \tag{2.30}$$

where $c = (\mathbf{V} \cdot \mathbf{H})$, $g^2 = n^2 + c^2 - 1$, and n is the index of refraction.

The geometrical attenuation factor is

$$G = \min\left\{1, \frac{2\,(\mathbf{N} \cdot \mathbf{H})\,(\mathbf{N} \cdot \mathbf{V})}{(\mathbf{V} \cdot \mathbf{H})}, \frac{2\,(\mathbf{N} \cdot \mathbf{H})\,(\mathbf{N} \cdot \mathbf{S})}{(\mathbf{V} \cdot \mathbf{H})}\right\} \tag{2.31}$$

There are several choices for the facet slope distribution function. Blinn [6] suggests three of them. The first corresponds to a Phong illumination model:

$$D_1 = b_1 \cos^{c_1} \alpha \tag{2.32}$$

where $\cos\alpha = (\mathbf{N} \cdot \mathbf{H})$. The second is the Gaussian one originally used in Torrance and Sparrow [75]:

$$D_2 = b_2 e^{-(c_2\alpha)^2} \tag{2.33}$$

The third is from [76]:

$$D_3 = b_3 \left[\frac{c_3^2}{\cos^2\alpha(c_3^2 - 1) + 1} \right]^2 \tag{2.34}$$

In all of these, the b's are arbitrary constants analogous to the k's in Phong illumination models. The c's (empirically) determine the width of the specular lobe. As Blinn [6] observes, if we define β to be the value of α at which a distribution drops to half its peak value, we have

$$\begin{aligned} c_1 &= -\frac{\ln 2}{\ln\cos\beta} \\ c_2 &= \frac{\sqrt{\ln 2}}{\beta} \\ c_3 &= \sqrt{\frac{\cos^2\beta - 1}{\cos^2\beta - \sqrt{2}}} \end{aligned}$$

Cook [15] considers an additional possibility originating with Beckmann and Spizzichino [4], which we will include here as

$$D_4 = \frac{1}{4m^2\cos^4\alpha} e^{-\left(\frac{1-\cos^2\alpha}{m^2\cos^2\alpha}\right)} \tag{2.35}$$

where m is the RMS slope of the surface. Unlike D_1 - D_3, there is no arbitrary b constant for this distribution. The relationship of m to the corresponding value of β is

$$m = \frac{\tan\beta}{\sqrt{\ln 2 - 4\ln\cos\beta}} \tag{2.36}$$

2.5.4.3 Neumann-Neumann Illumination Models

In [55], Neumann and Neumann discuss "separable" illumination models (i.e., those whose BRDF is of the form $a(\mathbf{S})\, r(\mathbf{V})$ for some functions a and r) and how their use

can speed up radiosity computation in non-diffuse environments. As an example, they describe a "lacquer model" of a purely diffuse material covered by a semi-transparent "lacquer" that absorbs but does not scatter light that passes through it. The resulting BRDF they derive is

$$f_r(\mathbf{S},\mathbf{V}) = c\exp\left\{-s\left(\frac{1}{(\mathbf{N}\cdot\mathbf{S})}+\frac{1}{(\mathbf{N}\cdot\mathbf{V})}\right)\right\} \tag{2.37}$$

where c and s are constants that characterize the model. We can make this comparable with Equations (2.24) and (2.29) by defining b_N as the value of f_r at $\mathbf{S}=\mathbf{V}=\mathbf{N}$. The equation then becomes

$$f_r(\mathbf{S},\mathbf{V}) = b_N\exp\left\{-s\left(\frac{1}{(\mathbf{N}\cdot\mathbf{S})}+\frac{1}{(\mathbf{N}\cdot\mathbf{V})}-2\right)\right\} \tag{2.38}$$

As we did before, we can relate s to a more geometrically meaningful quantity β that qualitatively measures the width of the specular peak. Keep the illumination normal ($\mathbf{S}=\mathbf{N}$) and increase the angle between $\mathbf{V}$ and $\mathbf{N}$ until f_r drops to half of its maximum (i.e. $\mathbf{V}=\mathbf{N}$) value. We define the resulting angle to be β. We can relate s to β:

$$s = -\frac{\ln 2}{1+\cos\beta} \tag{2.39}$$

Figure 2.5 shows what Neumann-Neumann illumination models (and the subsequently-discussed Minnaert illumination models) look like when applied to a sphere. Incident light for each illumination model is scaled to produce a peak unsaturated radiance at normal incidence ($\Phi_L = 0$) and then held constant as Φ_L and β are varied.

Note that as Φ_L increases, the image radiance decreases, unlike Phong illumination models. Also notice that the limb of the sphere ($(\mathbf{N}\cdot\mathbf{V})=0$) is always dark.

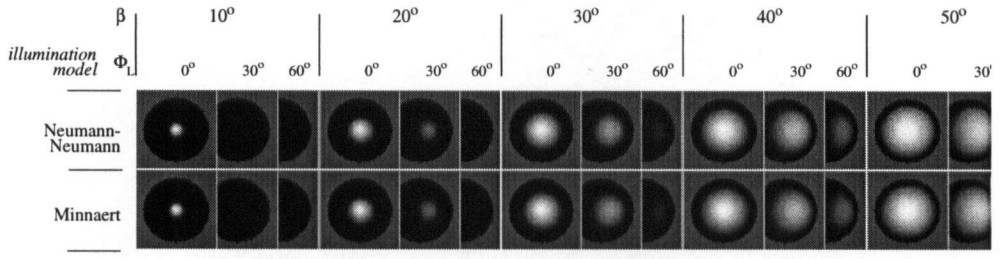

Figure 2.5: Sphere shaded with Neumann-Neumann and Minnaert illumination models for an assortment of incident angles with respect to the viewer (Φ_L) and specular distribution half-angles (β).

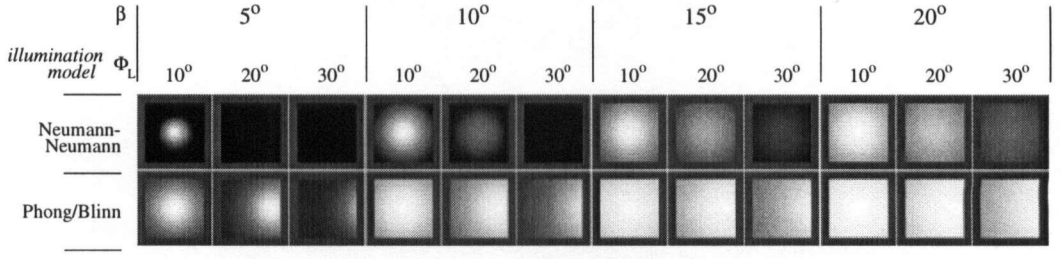

Figure 2.6: Comparison of a Square Shaded with a Neumann-Neumann Illumination Model and with a Phong Shader Using Blinn's F_s^B.

Neumann-Neumann illumination models exhibit peculiar behaviour when being applied to a specular surface. As Figure 2.5 shows, they do produce an acceptable specular peak. Nevertheless, for a given incident angle the resulting radiance always peaks when the viewer is looking in the normal direction. For a highly specular surface, we should expect the radiance to peak somewhat closer to the reflected direction. Instead, the peak occurs where the exponent of Equation (2.38) is minimized, which may not be the same direction.

Figure 2.6 illustrates this, comparing a Neumann-Neumann model with a specular Phong model using Blinn's F_s^B. To make sure the incident angle is constant,

we have replaced the sphere used in the test configuration of Figure 2.3 with a square whose normal is pointing at the observer. As the figure shows, increasing Φ_L causes the highlight to move to the right for the Phong model, as we would expect of a specular surface, but not for the Neumann-Neumann one.

This is not to say that Neumann-Neumann models do not apply to real-world phenomena, only that if they do, those phenomena do not resemble specularity.

2.5.4.4 Minnaert Illumination Models

In [52], Minnaert describes an illumination model derived from observations of the Moon. His original model is

$$L_r = b_M(\mathbf{N} \cdot \mathbf{S})^k(\mathbf{N} \cdot \mathbf{V})^{k-1}L_i \tag{2.40}$$

for some constants b_M and k. This corresponds to the BRDF

$$f_r(\mathbf{S}, \mathbf{V}) = b_M((\mathbf{N} \cdot \mathbf{S})\,(\mathbf{N} \cdot \mathbf{V}))^{k-1} \tag{2.41}$$

We can relate k to an angle β defined as in the previous section:

$$k = 1 - \frac{\ln 2}{\ln \cos \beta} \tag{2.42}$$

Figure 2.5 contrasts Minnaert illumination models with Neumann-Neumann illumination models. It is difficult to tell them apart. Their numerical values in these images differ by no more than 2%. (As a computational aside, since Minnaert illumination models are also separable, this suggests that a Minnaert illumination model should be able to take the place of a Neumann-Neumann illumination model with fewer arithmetic operations in most cases, especially if k is an integer.)

Minnaert illumination models exhibit the same peculiar behaviour as Neumann-Neumann illumination models when applied to a specular surface: for a fixed in-

cident angle, the resulting radiance peaks when the viewer looks in the normal direction.

2.5.5 Surface Models

The standard presentation of BRDF given in Section 2.4.3 leaves out an important fact: both of them may vary spatially, hence we should allow for[11]:

$$f_r \rightarrow f_r(\mathbf{P}, \mathbf{S}, \mathbf{V}). \tag{2.43}$$

We could deal with this somewhat by replacing the object being rendered with a collection of smaller objects with spatially-invariant surface properties, but this would cause the number of objects to increase considerably. Having to do this for a mottled surface such as a brick or an orange peel would inflate the number of objects to be rendered considerably and undesirably.

One way conventional renderers deal with spatial variation of the BRDF is by *texture mapping*, factoring it into spatial and directional components:

$$f_r \rightarrow t(u(\mathbf{P}), v(\mathbf{P}))f_r(\mathbf{S}, \mathbf{V}) \tag{2.44}$$

where the function t represents the *surface texture*: the spatial variation of reflective properties on a scale smaller than that of the object. $u(\mathbf{P})$ and $v(\mathbf{P})$ map spatial coordinates to *texture coordinates*. It is convenient to think of this as "wrapping" the surface with paper printed with a pattern that represents the texture. While this is a considerable reduction in flexibility from Equation 2.43, it is often adequate in practice.

[11] Even this is an oversimplification, as it assumes that light encountering the surface at point $\mathbf{P}$ is reflected from the surface at the same point. A more thorough treatment takes into account subsurface scattering and is treated by Hanrahan and Krueger [31] and Lalonde [41].

Another way renderers represent spatial BRDF as well as geometric variation is by allowing the surface normal used to compute f_r to vary with texture coordinates $u(\mathbf{P})$ and $v(\mathbf{P})$. This was first devised by Blinn [7] and is referred to as *bump mapping*. As the name suggests, bump mapping is particularly effective at representing surfaces whose reflective properties are the result of geometric variations such as roughness or machining as opposed to variations in material composition.

Work continues in this vein. Poulin and Fournier [60] extend BRDF modelling to include anisotropic reflection phenomena, and Fournier [21] describes how to model a given BRDF with a "normal distribution function".

2.5.6 Shading Models

There is a tendency to use the term "shading model" synonymously with "illumination model", but as Foley et al. [18] point out, a shading model may or may not make use of an illumination model. An illumination model is a description of what is actually happening in modelling or "world space". A shading model is a description of what is to be shown in viewing or "screen space". A shading model is what actually generates pixels. It is possible for a shading model to use an illumination model to generate a pixel, but it could also use a screen-oriented interpolation scheme to do so.

For instance, the Goraud shading model typically uses an illumination model to compute radiances at polygonal vertices, but then interpolates these in screen space during scan conversion. On the other hand, the Phong shading model (not to be confused with the Phong illumination model) computes normals at vertices, interpolates the normals in screen space during scan conversion, and (typically) applies an illumination model at each pixel.

An essential part of a shading model is the number of samples of an illumination model required per pixel. If this number is greater than one, as in the case of jittered or distribution raytracing, the samples must be filtered. If this number is less than one, the samples must be interpolated.

The use of a shading model is indicative of a perceived difference between what is necessary to model physically what is going on in the scene and what is necessary to generate an acceptable image. We will return to this idea in our discussion of radiosity in Section 2.5.7.4.

2.5.7 Rendering Techniques

The most distinctive part of a renderer is the rendering technique it uses. In this section, we will examine the predominant rendering techniques, ray tracing and radiosity, along with a new and promising one, photon maps.

2.5.7.1 The Rendering Equation

In his landmark work [38], Kajiya showed the unity of rendering techniques by connecting them to his form[12] of Equation (2.11). We can do much the same thing in the current context by noting that in Equation (2.12), in a state of thermodynamic equilibrium, L_i should be equal to $\mathcal{TB}L_r$. In other words, if we want a truly *global* solution, there should only be a single radiance distribution L which should satisfy

$$L = L_e + \mathcal{STB}L \tag{2.45}$$

which we rewrite as

$$L = \mathcal{G}L_e \tag{2.46}$$

[12]Like several other authors, Kajiya formulated Equation (2.11) in terms of "point-to-point" transfers, while we have chosen a "direction-to-point-to-direction" approach. The two are entirely equivalent.

where we have defined the *global illumination operator* $\mathcal{G}$ to be

$$\mathcal{G} = (\mathcal{I} - \mathcal{STB})^{-1} \tag{2.47}$$

where $\mathcal{I}$ is the identity operator. We can expand $\mathcal{G}$ as Kajiya does:

$$\mathcal{G} = \mathcal{I} + \mathcal{STB} + (\mathcal{STB})^2 + \cdots. \tag{2.48}$$

He then points out an interesting physical interpretation of this series. The first term represents light that is being given off by the surface on which we are computing L itself, the second term represents light from sources that directly illuminate the surface (a single "bounce" or surface interaction), and so on, so that the nth term represents light after $n-1$ "bounces". He also relates convergence of the series to energy conservation[13].

As Kajiya points out, rendering techniques approximate $\mathcal{G}$ in different ways. They also simplify the integral in Equation (2.11) (i.e., in $\mathcal{S}$) differently. We will discuss these distinctions in the following sections. Over time, these techniques have undergone hybridization[14] in order to improve their efficiency or functionality as they get incorporated into real renderers, but as we are more interested in their theoretical distinctions here, we will deal with each rendering technique in its "classical" form.

Due to space constraints, we will confine our discussion to currently-popular rendering techniques, omitting such work as that of Moravec [54], Weiler-Atherton [81], Warnock [79], and Roberts [62], which are now mainly of historical interest.

[13] Mathematically, this is equivalent to requiring that the spectral radius of the operator $\mathcal{STB}$ be less than unity.

[14] That's the nice name for it.

2.5.7.2 Depth-Buffering

By far the simplest rendering technique is depth buffering, also known as "z-buffering". First developed by Catmull [9] in 1974, the algorithm requires a buffer that contains a scalar value for each pixel. This value is the "depth" – the distance from the viewpoint to the nearest object that the pixel represents. Initially, all values in the depth buffer are (conceptually) infinite.

While primitives are being scan converted, the algorithm computes some measure of the distance from the object to the view plane for each pixel. In the case of opaque objects, this distance is compared with the depth buffer value. If (and only if) the new distance is smaller than the previous distance, the colour buffer is set to the appropriate value for the new primitive and the new distance is stored in the depth buffer.

With some limitations, non-refractive transparency is possible with a depth buffer. Williams [84] presents a way to use depth buffers to generate (hard) shadows. Since it places no constraints on the method used to compute the values that go into the colour buffer, depth buffering is compatible with all local illumination models.

Depth buffering can be implemented in hardware or software. The OpenGL graphics library ([40] and [8]) includes built-in support for depth-buffered rendering. Hardware depth buffering provides very fast rendering. OpenGL makes use of it if it is present and simulates it in software if it is not.

2.5.7.3 Raytracing

Classical ray tracing simplifies Equation (2.11) by setting f_r and f_t to be Dirac δ-functions (which we will describe below). This permits (mirror-like) reflection and refraction. The second way is by setting L_i to include a δ-function, which, as we will

see in Sections 2.5.1.1 and 2.5.1.2, permits point and directional luminaires. These are mutually exclusive since we cannot allow both the incident radiance and the BRDF or BTDF to be δ-functions.

Since each surface interaction involves samples of the BRDF and/or BTDF and possibly casting additional reflection/refraction rays, raytracing is compatible with all illumination models.

2.5.7.4 Radiosity

Although radiosity methods had been present in the mechanical engineering community for some time, they were first introduced to the computer graphics community by Goral, et al. [25] in 1984. In its classic form radiosity takes L to be isotropic and constant over a patch and therefore equal to $\frac{B}{\pi}$, where B is the radiosity of the patch. As derived in, for example, Cohen and Wallace [14], this leads to the classic radiosity equation:

$$(\mathbf{I} - \rho\mathbf{F})\,\mathbf{B} = \mathbf{E} \tag{2.49}$$

where, given N patches, $\rho\mathbf{F}$ is an $N \times N$ matrix whose elements are $\rho_i \mathbf{F}_{ji}$, ρ_i is the (isotropic) reflectivity of patch i, $\mathbf{F}_{ji}$ is the purely geometric *form factor*: the amount of energy leaving patch j that arrives at patch i, $\mathbf{B}$ is an N-element column vector of radiosities, and $\mathbf{E}$ is an N-element column vector of patch emissive exitances. Several conventional numerical linear algebra schemes have been applied to solve Equation (2.49), including Gauss-Seidel [25], Southwell (also known as "progressive refinement") [26], and multigrid [46].

Classical radiosity is compatible only with diffuse illumination models, but more recent research has extended this to glossy reflection. Immel et al. [35] developed a "global cube" algorithm that collected radiances instead of radiosity. This

approach produced realistic images, but was costly in both time and memory, especially for highly specular surfaces. Rushmeier and Torrance [64] developed the notion of "extended form factors" to deal with specular (including mirror) surfaces. Sillion et al. [69] deal with glossy reflection by representing both BRDFs and radiances with spherical harmonics.

Even more recently, Fournier [19] has shown how to incorporate non-diffuse phenomena by the use of separable BRDFs.

An important consideration in viewing the results of radiosity and other global illumination computations is the role of what Christensen, et al. [11] (citing Reichert [61]) refer to as the "final gather" step.

It takes place after Equation 2.49 or its equivalent has computed radiance distributions (or their equivalent) on all visible surfaces and amounts to a reevaluation of Equation 2.11 at the point in the scene intersected by a ray through the middle of each pixel. As Christensen, et al. [11] put it,

> Formally, this final gather corresponds to changing to a piecewise-constant basis, where the support of each basis function is the projection of a pixel onto a surface in the scene. This basis is tailored to be visually pleasing. The final gather smoothes the discontinuities in the wavelet representation and makes highlights, textures, and shadows crisper.

A final gather is effectively a local illumination calculation performed at the end of a global computation. It is an implicit acknowledgement of the fact that the criteria for physically-correct global illumination and a "visually pleasing" image are not identical.

2.5.7.5 Photon Maps

One of the most promising of recent rendering techniques are photon maps, developed by Jensen [37]. A photon map is a four-dimensional distribution of incident photons collected on a surface.

The first step of photon map global illumination is the generation of two photon maps built up by casting photons from all luminaires. The first photon map is a high-resolution caustics photon map made up of photons cast only in the direction of specular objects. The second photon map is made up of photons cast towards all objects.

The second step is the rendering step. In it, the photon maps of incident photons are combined with arbitrary BRDFs to produce reflected radiances made up of four terms: direct illumination, specular reflection, caustics, and soft indirect illumination. Each term is computed with appropriate photon maps and BRDF specular and/or diffuse components. As Jensen puts it:

> The photon map is used to generate optimized sampling directions, to reduce the number of shadow rays, to render caustics, and to limit the number of reflections traced as the scene is rendered with distribution ray tracing.

Lucifer, the algorithm we will present in Chapter 3, is coeval with photon maps. We will see that the two have much in common.

2.5.7.6 Comparison and Summary

This section will conclude with an analysis of the strengths and weaknesses of each of these models.

Effect	Chronological Order of Development → Depth Buffer	Ray Tracing	Radiosity	Photon Maps
diffuse surfaces	yes	yes	yes	yes
specular highlights	yes	yes	no	yes
transparency	yes	yes	no	yes
mirror reflection	no	yes	no	yes
refraction	no	yes	no	yes
sharp shadows	yes	yes	no	yes
soft shadows	no	no	yes	yes
diffuse lighting	yes	no	yes	yes
colour bleeding	no	no	yes	yes
caustics	no	no	no	yes

Table 2.1: Effects Available with Established Rendering Techniques

Table 2.1 lists a number of illumination effects and whether or not the various established rendering techniques are capable of them. One word of caution: If we evaluate this table for specific renderers, it becomes clear that it is an oversimplification. Some renderers which refer to themselves as raytracers, for instance, are capable of soft shadows. This is because those renderers have incorporated radiosity or other methods in hybrid form and are not "pure" raytracers in that sense. For this reason, Table 2.1 should not be taken to refer to specific renderers, only to rendering techniques.

Indeed, ray tracing and radiosity techniques can be enhanced (often by Monte Carlo methods) to produce a wider range of effects than Table 2.1 indicates. As in many other areas in which they are applied, however, Monte Carlo techniques exhibit slow convergence.

To be able to produce these and other effects, we will elaborate on a global illumination technique that allows a wider range of local illumination models.

Chapter 3

The Lucifer Algorithm

The algorithm we present here, which we refer to as the *Lucifer*[1] algorithm, is a sequential computational analogue of how energy distributes itself in a scene. It combines the physical concepts of isolation and energy conservation with a refineable spatial partitioning scheme. This work is a outgrowth of work done by Fournier, Fiume, Ouellette, and Chee [20] (hereafter referred to by its project name "*FIAT*").

3.1 Isolation

For possibly complex scenes, we can take a divide-and-conquer approach based on the principle of *isolation*, illustrated in Figure 3.1 (right).

Isolation is a conceptual tool: we place any part of our scene (including the viewer) within a volume V. Suppose then that somehow we can determine the distribution of radiance on the surface ∂V of V. Isolation says that for the purposes of solving for global illumination anywhere outside V, we can effectively replace the

[1] From the Latin for "light bearer".

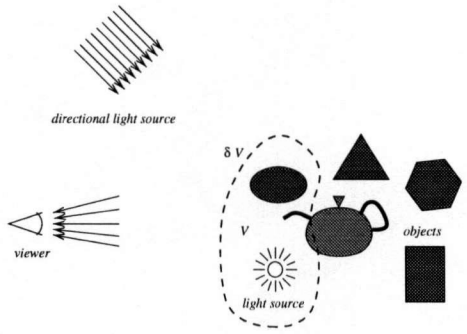

Figure 3.1: An Application of Isolation

contents of V with ∂V's radiance distribution.

Isolation gives us a way to deal with complex scenes. If we cannot deal with the whole scene, break it into volumes we *can* deal with as local illumination problems and transfer radiance distributions along the boundaries between the volumes.

3.2 Power Computation for Isolated Volumes

For any volume $\mathbf{V}_i$, physics demands energy conservation. In the steady state, this is equivalent to power conservation:

$$\Phi_{\text{in},i} + \Phi_{\text{em},i} = \Phi_{\text{out},i} + \Phi_{\text{abs},i} \tag{3.1}$$

where

$$\Phi_{\text{in},i} = \int_{\partial \mathbf{V}_i} \int_{\Omega^+_{\mathbf{N}}} L_{\text{in}} \, |\mathbf{N} \cdot \mathbf{S}| \, d\omega dA \tag{3.2}$$

is the flux entering $\mathbf{V}_i$,

$$\Phi_{\text{out},i} = \int_{\partial \mathbf{V}_i} \int_{\Omega^-_{\mathbf{N}}} L_{\text{out}} \, |\mathbf{N} \cdot \mathbf{S}| \, d\omega dA \tag{3.3}$$

is the flux leaving $\mathbf{V}_i$,

$$\Phi_{\mathrm{em},i} = \int_{\mathbf{V}_i} \int_{\Omega} j \, d\omega d\mathbf{V}_i + \int_{S_{\mathrm{em}}(\mathbf{V}_i)} \int_{\Omega^+_{\mathbf{N}}} L_e \ |\mathbf{N} \cdot \mathbf{S}| \ d\omega dA \tag{3.4}$$

is the flux generated within $\mathbf{V}_i$,

$\Omega^+_{\mathbf{N}}$ and $\Omega^-_{\mathbf{N}}$ at any point on ∂V are the unit hemispheres entirely outside and entirely inside $\mathbf{V}$, respectively, L_{in} is the radiance coming into $\mathbf{V}_i$ from $\Omega^+_{\mathbf{N}}$, L_{out} is the radiance passing out of $\mathbf{V}_i$ from $\Omega^-_{\mathbf{N}}$, Ω is the unit directional sphere, j is the volume emissivity within $\mathbf{V}$, L_e is the surface emissivity on all emissive surfaces $S_{\mathrm{em}}(\mathbf{V}_i)$ within $\mathbf{V}$, and $\Phi_{\mathrm{abs},i}$ is the amount of radiant power absorbed and not converted into radiant energy again (at least in $\mathbf{V}_i$).

3.3 Solution Order and Convergence

As with raytracing and radiosity, we are constructing a sequential analogue of what nature does in parallel. We need a sequential ordering. A reasonable way to proceed is to maintain the volumes in a queue sorted in order of decreasing *undistributed power* $\Phi_{\mathrm{in},i} + \Phi_{\mathrm{em},i}$ and to always concentrate our efforts on the volume $\mathbf{V}_i$ at the front of the queue. Once the undistributed power in this volume is distributed, the volume moves to the end of the queue. Note that, from Equations (3.3) and (3.4), undistributed power is computed from radiances, emissivities, and geometrical information about $\mathbf{V}_i$.

If, during the distribution process, we ensure that no more energy leaves $\mathbf{V}_i$ than was either incident upon or emitted within it, we can guarantee that the total undistributed power in the scene (i.e., in the queue) is monotonically non-increasing, since all the power distributed by $\mathbf{V}_i$ that does not get absorbed becomes incident on another volume or leaves the scene entirely, possibly reaching the observer.

One way to ensure that no excess energy is created is to require an energy-conserving local illumination model. We will present some necessary constraints for such a model in Section 4, where we will consider several illumination models commonly in use in computer graphics from the aspects of both energy conservation and Helmholtz reciprocity[2].

3.4 Spatial Partitioning

We require a tessellation of the scene that can be easily refined as needed as the solution progresses. Several data structures permit this, but the simplest one for our purposes is the octree.

As shown in Figure 3.2, we take octrees to be composed of cubic *cells*. The *root cell* contains the entire scene. Each cell is either a *parent cell* or a *leaf cell*. A parent cell has eight *child cells*. A leaf cell has no child cells. Only leaf cells are "volumes" in the sense of our previous discussion. Non-leaf cells are purely structural. Each cell except the root cell has seven *sibling cells*.

We also choose a minimum size for an octree cell. Cells that we cannot deal with that are this size will be solved trivially, but still conserving energy.

3.5 The *Lucifer* Algorithm

Figure 3.3 shows our rendering algorithm pseudocode. We start off with a scene s with a single cubic cell, *rootCell(s)*. We then create a priority queue q with, initially, a single element, *rootCell(s)*. q is always maintained in decreasing order of undistributed power.

[2]Note that *Lucifer* as presented here does not require reciprocity.

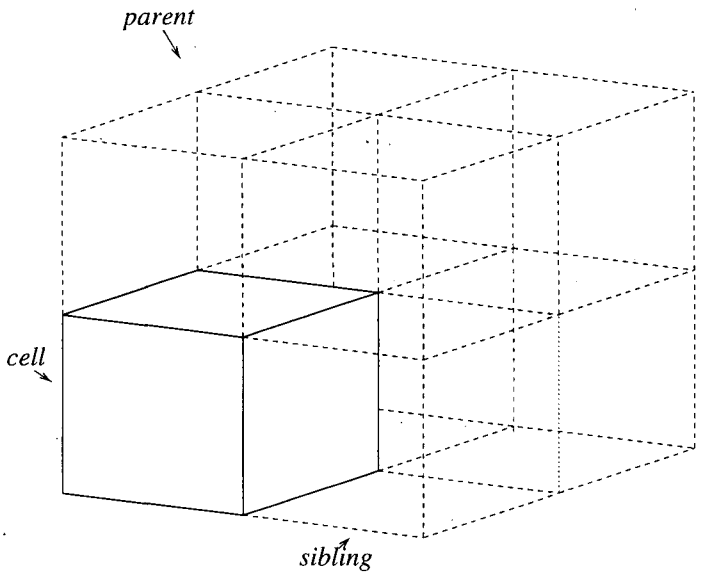

Figure 3.2: Octree Cell Nomenclature

If q is empty or *sumUndistPower(q)*, the sum of the undistributed powers of all cells in the queue, is lower than some prespecified *powerTolerance*, we consider the scene rendered.

Otherwise, we remove the cell with the largest amount of undistributed power, c, from the front of q. All further computation in the main loop is concerned only with c. This is why we refer to our algorithm as "light-driven": it is always concerned with the cell with the largest undistributed power.

We test c against the first of these cases that it matches:

- If it is empty, we call *propagate()* to transfer the radiance coming into c to its neighbors.
- If it is a cell that we know how to pass power through, we call *balance()*

```
render(scene s)
{
    queue q;
    cell c;

    q = createPriorityQueue(rootCell(s));
    while (sumUndistPower(q) > powerTolerance) {
        c = dequeue(q);
        if (isEmptyCell(c))
            propagate(c);
        else if (canDealWith(c))
            balance(c);
        else if (size(c) > minimumSize)
            subdivide(c, q);
        else
            trivialize(c);
    }
}
```

Figure 3.3: The *Lucifer* Algorithm

to perform the redistribution of the reflected and refracted radiance to c's neighbors.

- If it is above a specified minimum size, we call *subdivide()* to split c into eight child cells and add these back to q in order.

If it meets none of these criteria, we call *trivialize()* to perform an ad hoc solution that may be some combination of the first two cases, but which in any case guarantees that power balance is preserved.

propagate() is conceptually straightforward. It makes use of the invariance of radiance in the direction of propagation. Radiance at a given point in a given direction on an incoming cell wall defines a ray which may intersect an outgoing cell wall at a new point and direction. (The direction is, of course, the same, but may be expressed in a different coordinate frame.) We may alternatively project backwards from the outgoing cell wall to the incoming one.

balance() is somewhat more complicated. The ambiguity of the phrase "know how to pass power through" at this level of description is intentional. Our current interpretation of it is that the cell contains at most a single convex object. This allows us to decompose the procedure into four subtasks:

1. Transport radiance from the incoming walls to the object (or that part of the object which lies within the cell).

2. Interact the incoming radiance with the object's BRDF (and BTDF) to produce a reflected radiance being given off by the object.

3. Transport reflected and emitted radiance from the surface of the object to the outgoing walls.

4. Transport the unobstructed part of the incoming radiance to the outgoing walls.

Subtasks 1 and 3 are variations on *propagate()*, allowing transport to and from surfaces as well as cell walls. Subtask 2 requires surface interaction, as dictated by Equation (2.11). Subtask 4 is another variation on *propagate()*, including a geometric blocking function that inhibits propagation if the ray intersects the object within the cell. We call this variation *transblock()*.

3.6 Complexity

Let us compare the complexity of *Lucifer* and two rendering techniques that can produce comparable effects: raytracing and radiosity. Our measure of complexity is the mean time required per displayed pixel. We assume the scene has N_{obj} objects, of which N_{lum} are (initially) luminaires.

In classical raytracing, computing the intersection point of a ray and an object in the scene requires $O(N_{\text{obj}})$ time per ray[3]

If, to reduce aliasing effects, we oversample by a factor of N_{ray} rays per pixel, computing the primary (starting at the observer) intersections is $O(N_{\text{ray}}N_{\text{obj}})$. To determine shadows, we must cast N_{lum} shadow rays to light sources. As Whitted [83] first showed, we can treat reflective, refractive, and transmissive phenomena by constructing a tree at each intersection whose links correspond to reflected, refracted, and transmitted rays and whose nodes represent ray intersections with the scene. If we let D be the mean number of nodes in this tree (minimum of one), then noting that each ray, shadow or reflected/refracted, costs N_{obj}, the efficiency of classical raytracing is $O(N_{\text{ray}}N_{\text{obj}}(D+N_{\text{lum}}))$. As summarized by Arvo and Kirk in [1], many techniques have been devised to improve on this. With suitable restrictions on such renderer components as illumination model, luminaire model, and the spatial distribution of objects, considerable speedups are possible. Fujimoto et al. [23] cite rendering times that are practically independent of the number of objects in the scene (i.e., $O(N_{\text{ray}}D)$) but this is quite optimistic.

In classical radiosity, the computation of form factors is $O(N_{\text{obj}}^2)$ and while the inversion of the transfer matrix is in principle as high as $O(N_{\text{obj}}^3)$, it easily can be made $O(N_{\text{obj}}^2)$ by iterative and other means. Considerable speedups of radiosity are possible with hierarchical radiosity, as presented by Hanrahan et al. in [32]. Smits et al. in [71] extended this work by introducing clustering. As reported there, hierarchical radiosity has a complexity of $O(N_{\text{surf}}^2+N_{\text{patch}})$, where N_{surf} is the number of initial surfaces (proportional to N_{obj}) and N_{patch} is the final number of patches (which is at least a linear function of N_{surf}). Clustered radiosity, on the other hand,

[3]This is assuming a limiting depth – see Fiume [17] for additional details.

has a complexity of $O(N_{\text{surf}} \log N_{\text{surf}} + N_{\text{patch}})$. The linear or superlinear dependence of N_{patch} on N_{surf}. has given rise to a considerable amount of interest in optimal meshing schemes. Obviously, a clever adversary can defeat these speed-up schemes, but in practice the expected values apply.

In *Lucifer*, interactions are never between objects, but between a cell's walls and either the walls of neighboring cells or the object contained within the cell. Even though a particular cell may need to be balanced more than once, the number of times that needs to happen is dependent on the scene geometry, not on N_{cell}.

If the proportion f_{back} of power Ψ that comes back to the cell is fixed, then in order for the power to drop by a fractional tolerance ϵ_{pwr}, the cell must be balanced N_{bal} times, where

$$f_{\text{back}}^{N_{\text{bal}}} \Psi = \epsilon_{\text{pwr}} \Psi. \tag{3.5}$$

Taking the log of both sides, we find that N_{bal} is proportional to the log of ϵ_{pwr}.

Hence, as may be inferred from Figure 3.3 the asymptotic efficiency of *Lucifer* is $O(T_{\text{cell}} N_{\text{cell}})$ where T_{cell} is the cost of operation within a cell (a function of the maximum resolution used to express the radiance). Since our spatial partitioning scheme guarantees the number of objects, N_{cell} is $O(N_{\text{obj}})$, *Lucifer* is intrinsically $O(N_{\text{obj}})$ – asymptotically more efficient than other global illumination schemes.

It is important to note that in these techniques visibility is an important part (often the most important part) of the cost, and what makes it superlinear. *Lucifer* in a real sense "clusters" visibility, since it is carried along with the radiance representation from cell to cell. As an example, consider a light source blocked by a large object.

Before it reaches the blocker, the light flux is represented normally, and small shadows will be carried along as well. After it reaches the blocker, a large portion

of the light flux will be zero, the small shadows included in it will be absorbed. Any scheme capable of compressing the light flux data will take advantage of this, and objects and cells in the shadow will not even interact with the resulting light front.

To illustrate the trade-off consider the following scene. We have $N_{obj} \sim 10^7$ objects[4].

Any approach that has to consider pairs (either for explicit light transfer or for visibility) will have to deal with an order of 10^{14} such interactions.

To represent light fluxes at an acceptable level of detail, assume that the space is 10m by 10m by 3m, and we want about 1 cm accuracy. If we impose a maximum cell size of $1m^3$, that means we have have 300 cells to balance.

For each cell wall, we then require a positional resolution of 100 by 100 elements. If we assume the same resolution for direction (1/100th of a radian is approximately 0.57°), we need to represent 10^8 elements on each wall.

If we can use wavelet compression by 1000:1 (and we will see in Section 6.13 that this is not unreasonable), radiances on the walls can be represented with 10^5 coefficients. Even assuming that at each step of the propagation we deal with $O(N^2)$ interactions between elements (wall-to-object, wall-to-wall, and object-to-wall), that means $\sim 10^{10}$ such interactions. Multiplying by the number of cells, we get $\sim 3 \times 10^{12}$ interactions, which is a small fraction of the pairwise number, and this is not taking into account the speedups possible due to the hierarchical nature of the wavelet representation: If two coefficients do not interact, neither do any of their children.

The actual number of such computations is a function of how many times and at what level of detail we have to do this, but even at this level we have some hope of winning. An important point here is that the *Lucifer* numbers do not significantly

[4]We should note here that in the *Lucifer* context, the definition of the term "object" might be more powerful than in some other contexts.

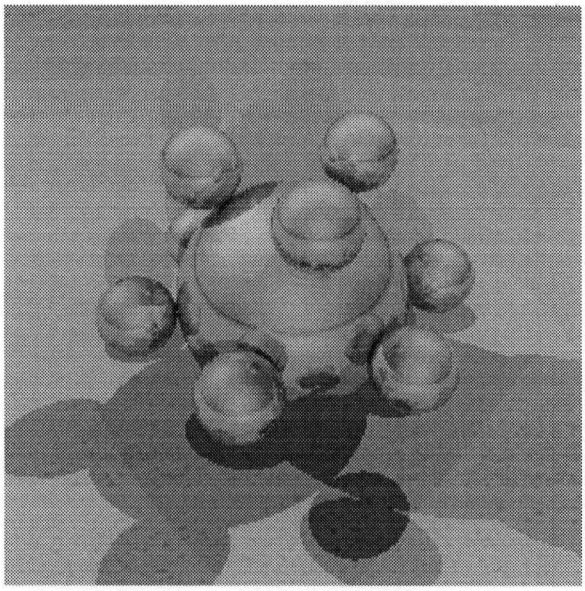

Figure 3.4: "Balls" Model

increase as a function of the number of primitives in the scene.

3.7 Light Through a Window

Let us consider an example of what radiance L on a cell wall looks like. L is a function of four variables (2 positional, 2 directional).

Figure 3.4 shows an instance of the "balls" model from Haines's Standard Procedural Database of test models for raytracers (described in [29]) rendered by a typical raytracing renderer. We have adapted this renderer to perform 4-dimensional ray tracing after a "light through a window" model. In this, we imagine light from a scene going through a unit square window. We further imagine sampling the radiance over the incoming hemisphere of directions on a uniformly spaced grid of positions. (In *Lucifer*, the "window" corresponds to a cell wall, but we'll discuss the

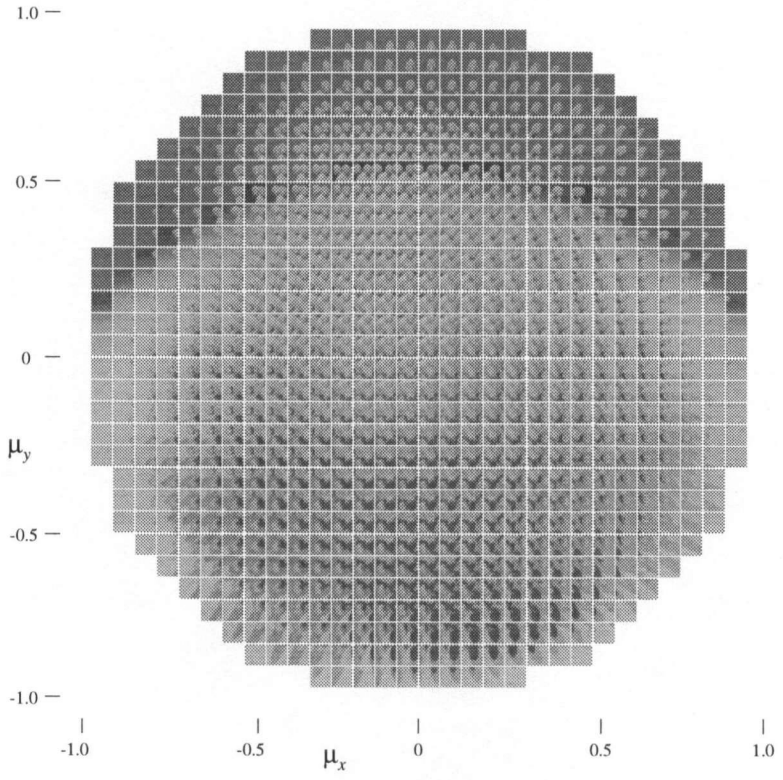

Figure 3.5: Array of Fixed Direction Views

more general case here.)

Before proceeding further, let us transform our definition of direction from a (θ, ϕ) parameterization to (μ_x, μ_y), where

$$\mu_x = \sin\theta \cos\phi \quad \mu_y = \sin\theta \sin\phi \tag{3.6}$$

This avoids discrepancies close to $\theta = 0$ and makes for a more uniform grid in Cartesian 3-space. $\mu_x^2 + \mu_y^2 = 1$ defines the *unit directional circle* (hereafter, UDC). All real directions must lie within this circle.

The modified renderer permits us to create an array of images, each image corresponding to a single direction (Figure 3.5) or a single position on the window

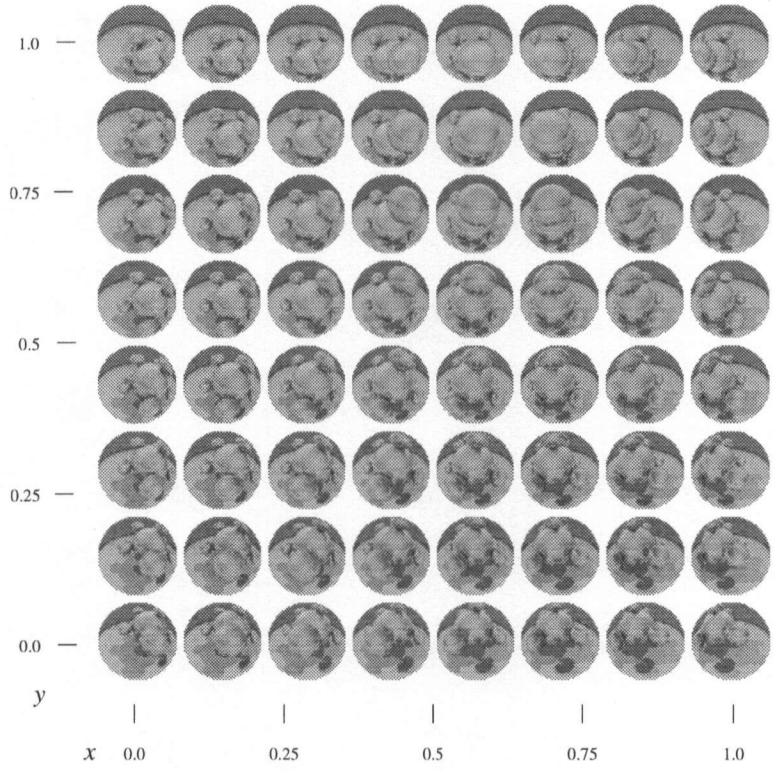

Figure 3.6: Array of Fixed Position Views

(Figure 3.6). In the former case, the individual images are parallel projections and in the latter case they are "fisheye" views. Note that, as in a perspective view, except in the special case of an orthographic (parallel) projection, a parallel projection of a sphere is not round.

From these examples, we can see that a 4-dimensional representation of radiance exhibits the same combination of discontinuities and relatively smooth areas we find in 2-dimensional images.

3.8 Radiance Representation

Radiance on a cell wall is a potentially discontinuous, generally non-analytic function of four variables (2 positional, 2 directional). We can represent radiance L at a point $\mathbf{P}$ on a cell wall or a surface and in a direction (μ_x, μ_y) with a finite element expansion with N_f degrees of freedom:

$$L(\mathbf{P}, \mu_x, \mu_y) = \sum_{i=1}^{N_f} a_i B_i(\mathbf{P}, \mu_x, \mu_y) \tag{3.7}$$

Even though the basis functions B_i may take on values for $\mu_x^2 + \mu_y^2 \geq 1$, we can disregard their behaviour there, since we never evaluate them in that region.

Choices for B_i include: box discretization, Fourier, discrete cosine, orthogonal polynomials, "light fields" (as described by Levoy and Hanrahan in [44]) and wavelets.

3.8.1 Box Discretization

For box discretization, as done by *FIAT*, the B_i are constant within a quantized direction for each quantized position.

Figure 3.7 shows the result of *Lucifer* using a box discretization on a simple model consisting of three squares forming a corner of a (root) cell with two spheres suspended within the cell. The square on the upper right is a diffuse white emitter and the other two squares are diffuse red (bottom) and gray (upper left) reflectors. One sphere is a mirror and the other is a diffuse green reflector. This image illustrates some of the effects possible with *Lucifer* – diffuse surfaces, colour bleeding, soft shadows, and mirror reflection – all produced with the same rendering technique.

There are various ways to perform propagation and object interaction of the discretized radiances. *FIAT* describes one possible way, equivalent to the one

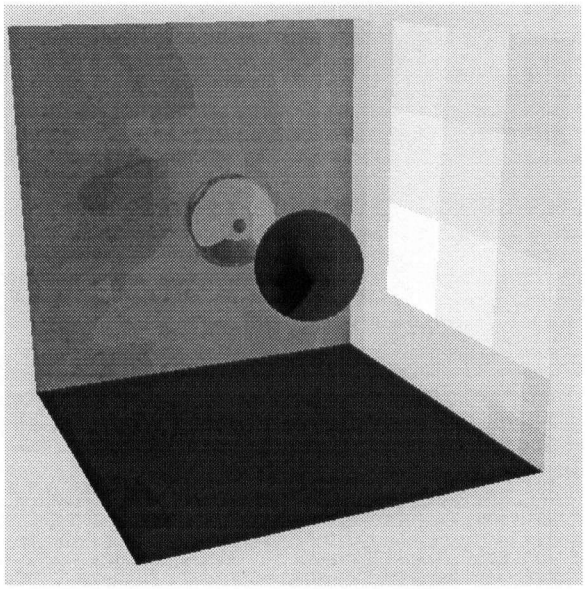

Figure 3.7: Box Discretization Example

we used for Figure 3.7. They have a common shortcoming, however: by quantizing angles and positions, box discretization causes various rendering artifacts. These can be reduced by increasing the number of quantized positions or directions, but this causes memory requirements to increase dramatically, even with dynamic allocation of memory.

In practice, the CPU time and memory required for box discretization is limiting. Figure 3.7, for example, required about 9.3MB of memory and 3.2 CPU-hours on an SGI Indigo 2 workstation.

3.8.2 Wavelets

Wavelets are a promising alternative to other basis functions. They were successfully applied to radiosity solutions by Gortler, et al. [28] and Schröder, et al. [65]. Schröder and Hanrahan [66] and Christensen, et al. [12] extended this work from

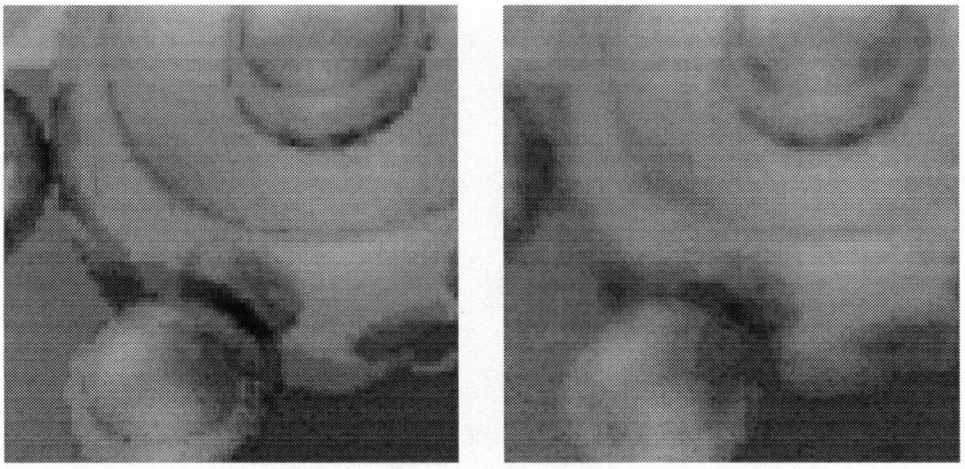

Figure 3.8: Original Image (left) and Its Reconstruction from 4-Dimensional Wavelet Coefficients (right)

radiosity to the representation of radiance itself. In this section, we will consider the general applicability of such a wavelet representation of radiance to *Lucifer*.

Appendix A describes some of the properties of one-dimensional wavelets. There are two properties of particular interest for radiance representation. First is their ability to approximate L^2 functions, even those with discontinuities, with a relatively sparse set of coefficients.

As an example of this, we apply wavelet compression to the 4-dimensional data shown in Figure 3.5. On the left of Figure 3.8 is a magnified view of one image from Figure 3.5 in a particular direction, on the right is a view in the same direction that was reconstructed from a 4-dimensional wavelet transform of the original data compressed by 96% – only the top 4% (in magnitude) of the original coefficients were retained. It is important to note that this compression was in all four dimensions

of the data *prior* to reconstruction.

There is an infinite number of base scaling functions $\phi(x)$ and therefore an infinite number of possible wavelets. For the example above, we used the $L = 2$ "Coiflet" wavelet described in [16], for these, but other wavelets gave qualitatively similar results. Part of our work will be to find which wavelet basis works best for radiance representation. Most importantly, we need to evaluate the tradeoff between improving approximation by increasing N_v, the number of vanishing moments, and minimizing operation count by reducing W_h, the size ("support") of $\{h_m\}$, the set of wavelet coefficients.

Other considerations in selecting a wavelet include:

- Is $\phi(x)$ symmetric about some value of x?
- Does $\phi(x)$ need to have an analytical form? (This leads to biorthogonal wavelets.)
- Does refinement of the wavelet interpolate the coarser values?
- What is the tradeoff between reducing W_h and representing discontinuities compactly?

The second property of wavelets relevant to *Lucifer* is their dyadic nature, which corresponds well with our octree spatial subdivision scheme. Splitting a cell whose walls represent radiance with with wavelets amounts to a partitioning of the wavelet coefficients.

We will consider these issues further in Chapter 5.

Chapter 4

Making Illumination Models More Physically Plausible

In our presentation of the Lucifer algorithm in Chapter 3, particularly in Section 3.3, we described the need for illumination models that conserved energy. We will describe such models in this chapter. This work has been published separately in reference [45] and (more accessibly) in reference [47].

Illumination modelling computation is an essential part of any rendering algorithm. Getting an exact physical model of the interaction of light with a surface is, for most surfaces occurring in the real world, a very difficult problem. Consequently, much effort has been expended on finding approximations that are both good-looking and quickly computed. An extensive, if somewhat dated, summary of these illumination models is in Hall [30].

Looking good and being quickly computable are sufficient criteria for most raytracing renderers (see Section 2.5.7.3). If one of these illumination models is used in a radiosity computation, however, it is necessary to additionally ensure that, since

radiosity is based on principles of energy conservation (see Kajiya [38], for example), the illumination model itself conserves energy. The global illumination model we present in Chapter 3 also requires energy conservation.

Another recent technique described in Larson (ne' Ward) [78] constructs functions to fit actual BRDFs. Such fits are likely to be more successful if the functions themselves are, like the data they fit (as demonstrated empirically in Clarke and Perry [13]), consistent with physics (at least within experimental uncertainty).

So a need has arisen for illumination models that are not only good-looking and easy to compute, but are also "physically plausible". [1]

In this chapter we will look at the correspondence between the fundamental reflectance equation (2.9) and traditional illumination models. Then we will look at ways to modify those models to make them more physically plausible.

4.1 Energy Conservation

The first physical constraint we will examine with respect to illumination models is that of energy conservation. Physically plausible illumination models must obey energy conservation. In a steady-state scene, the input power and the rest of the scene configuration (objects and their positions, surface properties, etc.) are taken not to vary with time. In such a situation, energy conservation is synonymous with power conservation. The total amount of power reflected, i.e., $M\,dA$, where M is the exitance, must be less than or equal to the total power incident $E\,dA$, where E

[1] We use the term *plausible* here in contrast to that of *feasible* in Neumann and Neumann [55]. A "feasible" illumination model is one that we can imagine constructing physically. This is not always possible. The weaker definition of a "plausible" illumination model is one whose existence does not violate physics.

is the irradiance. This must hold for all areas dA in our scene, hence

$$M \leq E \tag{4.1}$$

From Siegel and Howell [68], we have an equation similar to Equation (2.7) describing the exitance dM due to a reflected radiance L_r radiated into an infinitesimal solid angle $d\omega_r$ around a direction $\mathbf{V}$, so that

$$dM = (\mathbf{N} \cdot \mathbf{V})\, L_r\, d\omega_r \tag{4.2}$$

We substitute Equation (2.9) into this and integrate over $\Omega_{\mathbf{N}}^{+}$ to get

$$M = \int_{\Omega_{\mathbf{N}}^{+}} \int_{\Omega_{\mathbf{N}}^{+}} f_r(\mathbf{S}, \mathbf{V})\, L_i\, (\mathbf{N} \cdot \mathbf{S})\, (\mathbf{N} \cdot \mathbf{V})\, d\omega_i\, d\omega_r \tag{4.3}$$

We can also integrate Equation (2.7) to get

$$E = \int_{\Omega_{\mathbf{N}}^{+}} L_i\, (\mathbf{N} \cdot \mathbf{S})\, d\omega_i \tag{4.4}$$

So, if we make the trivial assumption that $E > 0$, we divide both sides of Equation (4.1) by E to get

$$\frac{\int_{\Omega_{\mathbf{N}}^{+}} \int_{\Omega_{\mathbf{N}}^{+}} f_r(\mathbf{S}, \mathbf{V})\, L_i\, (\mathbf{N} \cdot \mathbf{S})\, (\mathbf{N} \cdot \mathbf{V})\, d\omega_i\, d\omega_r}{\int_{\Omega_{\mathbf{N}}^{+}} L_i\, (\mathbf{N} \cdot \mathbf{S})\, d\omega_i} \leq 1 \tag{4.5}$$

Energy conservation does not depend upon the particular L_i distribution. Given any L_i, Equation (4.5) must hold, so, as we did with the Phong illumination models in Section 2.5.4.1, let us use a directional light source of the form

$$L_i = E_N\, \delta(\cos\theta_i - \cos\theta_i)\, \delta(\phi_i - \phi_i) \tag{4.6}$$

to represent a directional source of normal irradiance E_N in a direction $\mathbf{S}$. According to the ANSI/IES standard [36], M/E in this case becomes the "directional-hemispherical reflectance", which we will refer to as k_ρ[2]. Integrating the δ-functions

[2] In Neumann and Neumann [55], this is referred to as "albedo", but that usage is imprecise as the definition of that term does not require a unidirectional source. In addition, "albedo" is not defined in the standard.

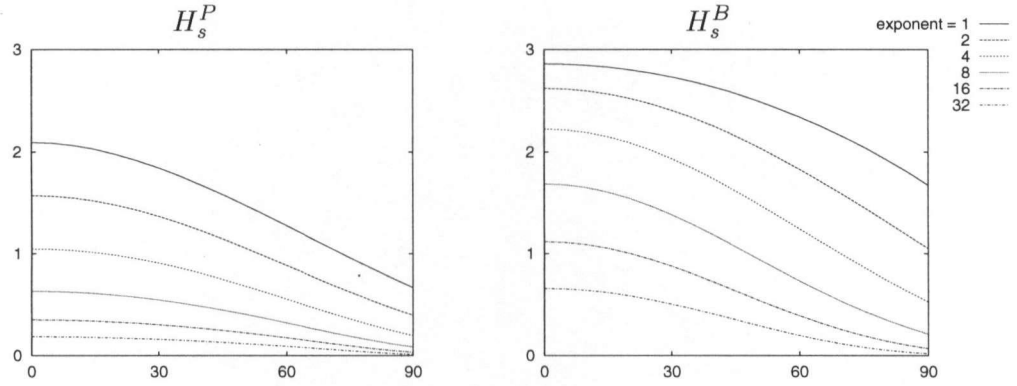

Figure 4.1: Specular Integrals $H_s(\mathbf{S})$ for Phong's F_s^P (left) and Blinn's F_s^B (right)

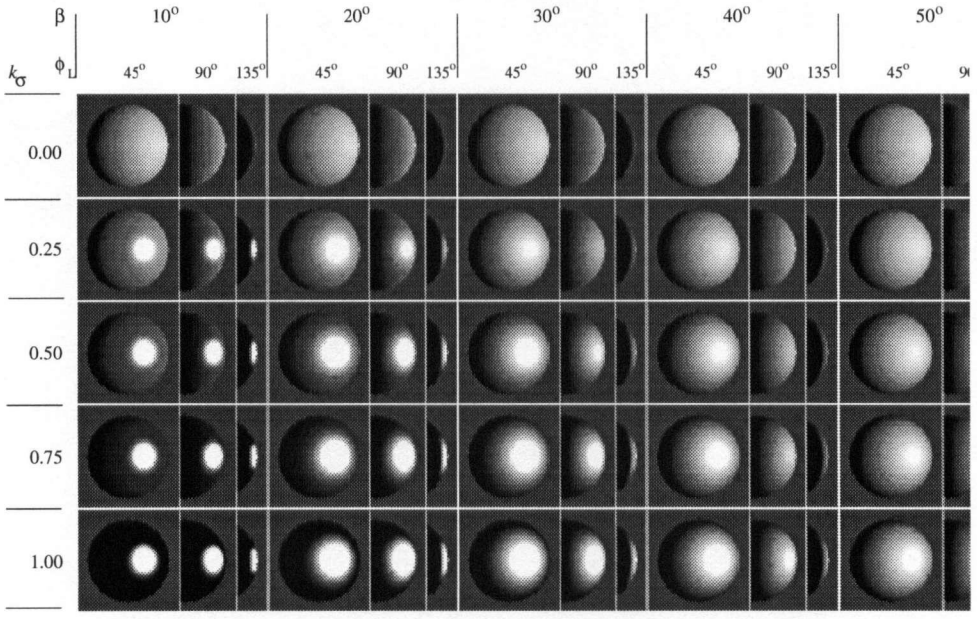

Figure 4.2: Sphere shaded with an energy-conserving Phong illumination model, using Blinn's F_s^B, for an assortment of incident angles with respect to the viewer (Φ_L), specular fractions (k_σ), and specular distribution half-angles (β).

and cancelling out common factors, we get

$$k_\rho = \int_{\Omega_{\mathbf{N}}^+} f_r(\mathbf{S}, \mathbf{V})\,(\mathbf{N} \cdot \mathbf{V})\,d\omega_r \leq 1 \tag{4.7}$$

4.1.1 Making Phong Illumination Models Conserve Energy

Let us apply these results to a Phong illumination model to see what constraint(s) energy conservation leads to. Recalling Equation (2.27), Equation (4.3) becomes

$$M = E_N \left[k_d \, \pi \, (\mathbf{N} \cdot \mathbf{S}) + k_s \, H_s(\mathbf{S}) \right] \tag{4.8}$$

where we have defined

$$H_s(\mathbf{S}) = \int_{\Omega_{\mathbf{N}}^+} F_s(\mathbf{S}, \mathbf{V}) \, (\mathbf{N} \cdot \mathbf{V}) d\omega_r \tag{4.9}$$

Figure 4.1 shows H_s evaluated numerically using both Equations (2.20) and (2.21) for a variety of specular exponents. Note that H_s is a function of the incident direction and specular exponent only and that it can be thought of as an integral operator acting on a given F_s.

As we might expect, Equation (4.4) becomes

$$E = E_N \, (\mathbf{N} \cdot \mathbf{S}) \tag{4.10}$$

so that

$$k_\rho = k_d \, \pi + k_s \frac{H_s(\mathbf{S})}{(\mathbf{N} \cdot \mathbf{S})} \tag{4.11}$$

To guarantee energy conservation regardless of reflection geometry, it is necessary to guarantee that $k_\rho \leq 1$ for all incident directions. But there is a problem here. Given the F_s's in Equations (2.20) and (2.21) and regardless of $\mathbf{S}$, it is always the case that $F_s \geq 0$ and, furthermore, there is always some non-vanishing region of $\Omega_{\mathbf{N}}^+$ over which $F_s > 0$. That means that H_s is always > 0, as Figure 4.1 illustrates. So that if $k_s > 0$, it is always possible to choose θ_i close enough to 90^o that k_ρ will be greater than one. We therefore conclude that the specular terms of Phong illumination models do not conserve energy at sufficiently large incident angles.

After Neumann and Neumann [55], let us consider a different formulation of an illumination model. Start from Equation (2.19), but suppose that, instead of being constant, we allowed k_s to vary with $\mathbf{S}$ in such a way that energy conservation was maintained. (As Equation (4.11) shows, we are not getting any trouble from the diffuse term, so we will leave it alone.)

Let k_σ be the fraction of exitance that is reflected specularly:

$$k_\sigma \equiv \frac{\int_{\Omega_{\mathbf{N}}^+} dM_{spec}}{M} = \left(1 + \frac{k_d\,\pi\,(\mathbf{N}\cdot\mathbf{S})}{k_s\,H_s(\mathbf{S})}\right)^{-1} \tag{4.12}$$

We can solve Equations (4.11) and (4.12) for k_d and k_s to get

$$k_d = \pi^{-1}\,k_\rho\,(1 - k_\sigma) \tag{4.13}$$

$$k_s = \frac{k_\rho\,k_\sigma\,(\mathbf{N}\cdot\mathbf{S})}{H_s(\mathbf{S})}$$

so we can rewrite the BRDF for the new illumination model as

$$f_r(\mathbf{S},\mathbf{V}) = k_\rho\left[\frac{1-k_\sigma}{\pi} + \frac{k_\sigma\,F_s(\mathbf{S},\mathbf{V})}{H_s(\mathbf{S})}\right] \tag{4.14}$$

We can also construct the analogue of Equation (2.19) to express this result in terms of radiance:

$$L_r = k_\alpha L_a + k_\rho\,(\mathbf{N}\cdot\mathbf{S})\left[\frac{1-k_\sigma}{\pi} + \frac{k_\sigma\,F_s(\mathbf{S},\mathbf{V})}{H_s(\mathbf{S})}\right] E_d \tag{4.15}$$

where

$$k_\alpha = k_\rho\left[1 + k_\sigma\left(\int_{\Omega_{\mathbf{N}}^+} (\mathbf{N}\cdot\mathbf{S})\frac{F_s(\mathbf{S},\mathbf{V})}{H_s(\mathbf{S})}d\omega_i \;-\; 1\right)\right] \tag{4.16}$$

corresponds to Equation (2.28).

Figure 4.2 shows what such an illumination model looks like when applied to a sphere with a single directional light source and no ambient radiance. For this

figure, we have used Blinn's F_s^B. We have also taken $k_\rho = 1$, since any other value would just be a uniform reduction by a constant factor in image radiance. Notice that, unlike Figures 2.4 and 2.5, the highly specular parts of the printed images are necessarily clamped in order to show the diffuse parts.

4.1.2 Do Torrance-Sparrow Illumination Models Conserve Energy?

Figure 4.3 shows some numerical integrations of Equation (4.7), contrasting Phong illumination models with Torrance-Sparrow illumination models. All Torrance-Sparrow illumination models were computed with a Fresnel factor $F = 1$ (i. e. a large index of refraction) to show the worst case.

One way to produce an energy-conserving Torrance-Sparrow illumination model suggests itself: simply choose any value of b_j such that

$$b_j < \frac{1}{\left[\frac{k_\rho^{Tj}}{b_j}\right]_{\max}} \tag{4.17}$$

where $\left[\frac{k_\rho^{Tj}}{b_j}\right]_{\max}$ is the maximum value as shown in Figure 4.3. (The Beckmann-Spizzichino distribution is not a problem as long as its integral is always less than unity, and it has no b-coefficient to adjust anyway.)

Nevertheless, doing this would probably be a mistake. To see why, look at the plot for k_ρ^{T1}/b_1, the Torrance-Sparrow illumination model with the Phong microfacet distribution. Notice that it does not diverge as $\theta_i \to 90^o$, even though k_ρ^P/k_s, the corresponding Phong illumination model with a Phong specular term, *does* diverge. The same is true for k_ρ^{T2}/b_2 compared to k_ρ^B/k_s.

Why should this be? The answer lies in the geometrical attenuation factor G. As $\theta_i \to 90^o$, G is guaranteed to be less than or equal to unity and, if $(\mathbf{V} \cdot \mathbf{H}) > 0$ (i.e., $\mathbf{V}$ and $\mathbf{S}$ are not antiparallel), it will vanish in the limit.

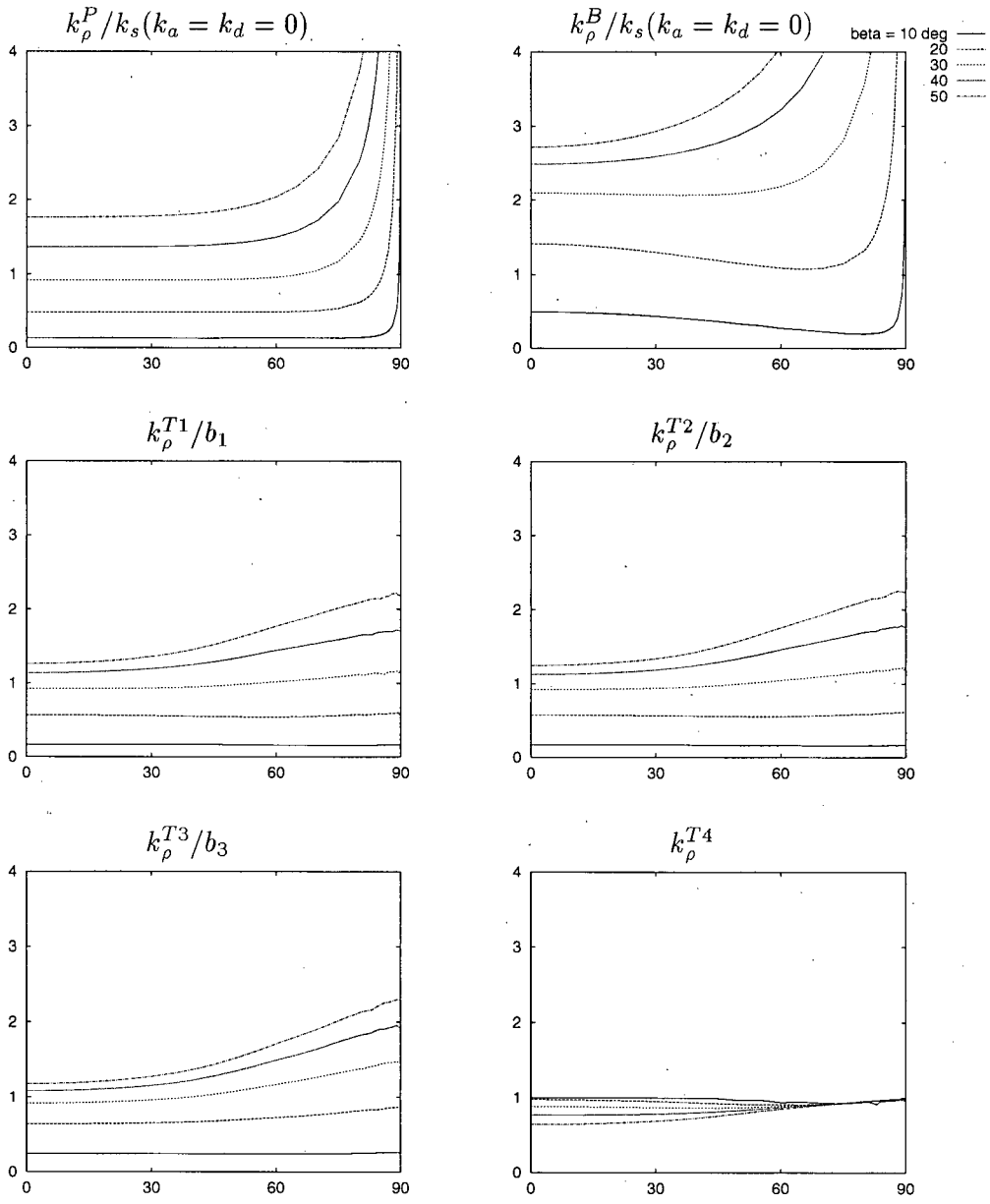

Figure 4.3: Directional-hemispherical reflectances as functions of incident angle for a Phong illumination model (k_ρ^P), a Blinn illumination model (k_ρ^B), and Torrance-Sparrow illumination models with Phong (k_ρ^{T1}), original Torrance-Sparrow (k_ρ^{T2}), Trowbridge (k_ρ^{T3}), and Beckmann-Spizzichino (k_ρ^{T4}), microfacet distributions.

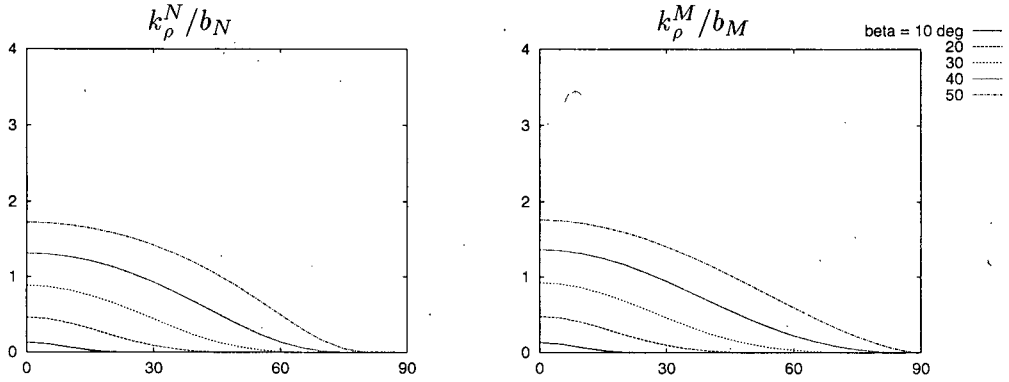

Figure 4.4: Directional-hemispherical reflectances as functions of incident angle for a Neumann-Neumann illumination model (k_ρ^N) and a Minnaert illumination model (k_ρ^M)

But what does this really mean? If we go back to the derivation of the geometrical attenuation factor in Torrance and Sparrow [75], we see that G is designed to compensate for the blocking of light that falls on a facet and the masking of light that the facet reflects. The blocking and masking agents are themselves other facets.

This leads to a critical question for Torrance-Sparrow illumination models and energy conservation: What happens to the light that gets blocked or masked? The illumination model does not treat secondary reflection. Instead, it acts as though the blocked or masked light were completely absorbed by the surface. This is unlikely.

For this reason, while it may be reasonable to consider the use of Torrance-Sparrow illumination models as *ad hoc* basis functions to fit empirical data, as was done in Larson [78], we should do so realizing that it is not really "fair" to use Torrance-Sparrow illumination models in an energy-conserving context. Basis functions that properly account for blocked and masked light are needed, but we will not attempt to derive them here.

4.1.3 Making Neumann-Neumann and Minnaert Illumination Models Conserve Energy

Figure 4.4 shows some numerical integrations of Equation (4.7) for Neumann-Neumann and Minnaert illumination models. Contrast these with those of Figure 4.3.

Like the Torrance-Sparrow illumination models, k_ρ is bounded in both cases, so we can put a limit on b_N or b_M to assure energy conservation.

In the case of a Minnaert illumination model, we can go a bit further and note that k_ρ can be determined analytically (using Equation (2.27), as was done in Woodham and Lee [85]). The resulting BRDF can be formulated directly in terms of k_ρ:

$$f_r(\mathbf{S},\mathbf{V}) = k_\rho \frac{(k+1)}{2\pi}((\mathbf{N}\cdot\mathbf{S})\,(\mathbf{N}\cdot\mathbf{V}))^{k-1} \tag{4.18}$$

where, as always, any value of k_ρ between 0 and 1 will guarantee energy conservation.

4.2 Making Illumination Models Reciprocal

The second physical constraint we will examine with respect to illumination models is that of Helmholtz reciprocity. A physically plausible illumination model ought to obey Helmholtz reciprocity (see Siegel and Howell [68]). In terms of the BRDF, this means that

$$f_r(\mathbf{S},\mathbf{V}) = f_r(\mathbf{V},\mathbf{S}) \tag{4.19}$$

for all $\mathbf{V}$ and $\mathbf{S}$ in $\Omega_{\mathbf{N}}^{+}$.

4.2.1 Are Phong Illumination Models Reciprocal?

Using the BRDF of a Phong illumination model given in Equation (2.20), and expressing F_s in the functional form $F_s(\mathbf{S},\mathbf{V})$, we see that such an illumination

model will be reciprocal if

$$\frac{F_s(\mathbf{S},\mathbf{V})}{(\mathbf{N}\cdot\mathbf{V})} = \frac{F_s(\mathbf{V},\mathbf{S})}{(\mathbf{N}\cdot\mathbf{S})} \tag{4.20}$$

Substitution of both F_s^P from Equation (2.20) and F_s^B from Equation (2.21) reveals that neither of these illumination models is reciprocal[3].

Is our energy-conserving modified Phong illumination model reciprocal? Applying Equation (4.19) to Equation (4.14), we are asking if

$$\frac{F_s(\mathbf{S},\mathbf{V})}{H_s(\mathbf{S})} = \frac{F_s(\mathbf{V},\mathbf{S})}{H_s(\mathbf{V})} \tag{4.21}$$

Again, the answer is no for both F_s's.

4.2.2 Are Torrance-Sparrow Illumination Models Reciprocal?

By inspection, it is easy to see that the Torrance-Sparrow illumination models are all reciprocal. This should come as no surprise, as the assumption of reciprocity was part of their derivation in [75]. Unfortunately, the arguments made above about their energy conservation still limits their plausibility.

4.2.3 Are Separable Illumination Models Reciprocal?

It is also easy to see by inspection that both Neumann-Neumann and Minnaert illumination models are reciprocal. Again, this is because reciprocity was part of their derivation.

As with Torrance-Sparrow illumination models, separable illumination models could be used as *ad hoc* basis functions. Care needs to be taken, though, to retain reciprocity. Given two separable BRDFs $f_{r1}(\mathbf{S},\mathbf{V}) = a_1(\mathbf{S})r_1(\mathbf{V})$ and $f_{r2}(\mathbf{S},\mathbf{V}) = a_2(\mathbf{S})r_2(\mathbf{V})$, a simple linear combination of the form $f_r(\mathbf{S},\mathbf{V}) = c_1 f_{r1}(\mathbf{S},\mathbf{V}) + c_2 f_{r2}(\mathbf{S},\mathbf{V})$ for some constants c_1 and c_2 is not, in general, reciprocal.

[3] Even though $F_s^B(\mathbf{S},\mathbf{V}) = F_s^B(\mathbf{V},\mathbf{S})$. and $F_s^P(\mathbf{S},\mathbf{V}) = F_s^P(\mathbf{V},\mathbf{S})$.

We can, however, generalize the work of Westin, et al. [82] and note that if we can express the BRDF in a separable matrix product form

$$f_r(\mathbf{S}, \mathbf{V}) = [\mathbf{Q}(\mathbf{S})]^T \mathbf{M}\mathbf{Q}(\mathbf{V}) \tag{4.22}$$

where $\mathbf{Q}(\mathbf{x})$ is some (possibly non-linear) vector function of $\mathbf{x}$ and $\mathbf{M}$ is a constant matrix, then this BRDF is reciprocal for any symmetric $\mathbf{M}$. This is how we could combine multiple Neumann-Neumann, Minnaert, or other similar basis functions: set them to be the elements of $\mathbf{Q}$ and find a symmetric $\mathbf{M}$ to fit our data, as Westin, et al. did for spherical harmonics.

4.3 An Energy-Conserving, Reciprocal Illumination Model

Objections can be raised to all of the illumination models we are presented so far, either on the grounds of implausibility (Phong) or of behaviour that, while plausible, is unlikely to fit a real BRDF (Torrance-Sparrow, Neumann-Neumann, Minnaert).

Consider instead a Phong illumination model formulated like Equation (2.24), but using Blinn's F_s^B and omitting the $(\mathbf{N} \cdot \mathbf{S})$ in the denominator of the specular term, we find

$$f_r(\mathbf{S}, \mathbf{V}) = k_d + k_s F_s^B(\mathbf{S}, \mathbf{V}) \tag{4.23}$$

Obviously, since F_s^B is reciprocal, this BRDF is reciprocal. (We could also have done this with F_s^P, since it is also reciprocal.)

Figure 4.5 shows the resulting k_ρ. It is bounded, so we can always conserve energy by limiting k_s and k_d. (Unfortunately, we cannot formulate the illumination model in terms of k_ρ and k_σ as we did above, since doing this makes the illumination model non-reciprocal.)

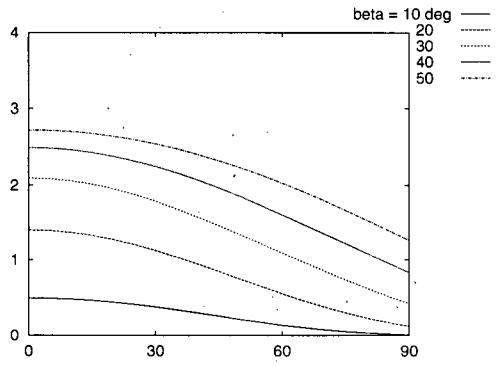

Figure 4.5: Directional-hemispherical reflectance as a function of incident angle for a reciprocal Phong illumination model, using Blinn's F_s^B (k_ρ^H)

Illumination Model	Plausibility: Conserves Energy?	Plausibility: Reciprocal?	Other Objections
Phong	no	no	
Energy-Conserving Phong	yes	no	
Torrance-Sparrow	yes	yes	no secondary reflection
Neumann-Neumann	yes	yes	non-specular behaviour
Minnaert	yes	yes	non-specular behaviour
Reciprocal Phong-Blinn	yes	yes	

Table 4.1: Summary of Plausibility Results

Figure 4.6 shows some images produced with a reciprocal Phong illumination model. As in Figure 2.4, k_s varies between 0 and 1 and k_d is taken to be $1 - k_s$.

While resembling Figure 2.4, the images for large Φ_L are dimmer, as we might expect from the absence of the $(\mathbf{N} \cdot \mathbf{S})$ in the specular denominator. Nevertheless, they are not as diminished as those of the separable illumination models in Figure 2.5 (which does not even bother showing $\Phi_L > 60^o$).

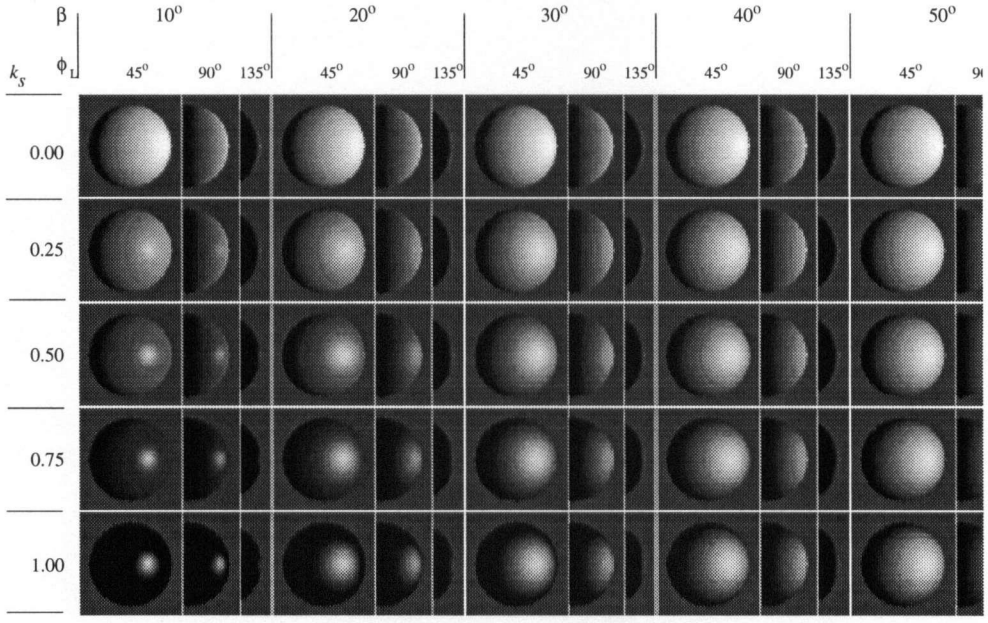

Figure 4.6: Sphere shaded with a reciprocal Phong illumination model for an assortment of incident angles with respect to the viewer (Φ_L), specular coefficients (k_s), and specular distribution half-angles (β)

4.4 Summary of Plausibility Results

We have examined a number of illumination models commonly used in graphics, looking at their plausibility in terms of energy conservation and reciprocity. Our results are summarized in Table 4.1.

As originally defined, Phong illumination models fail on both counts. It is possible to modify a Phong illumination model to conserve energy and even, as shown in Equation (4.14), have an energy-based parameterization, but this rules out satisfying reciprocity.

Torrance-Sparrow illumination models are reciprocal and appear to conserve energy, but their underlying derivation fails to account for blocked and masked

energy. They may still be useful, however, as *ad hoc* basis functions.

Neumann-Neumann and Minnaert illumination models are similar. Both are plausible: they conserve energy and are reciprocal. Minnaert illumination models have been used successfully to fit radiometric data. While it would be worth trying one of them as a basis, we expect that they will prove less useful with highly specular surfaces because both illumination models peak undesirably in the normal direction.

A differently-modified Phong illumination model given in Equation (4.23) is reciprocal and can be constrained to conserve energy.

Chapter 5

Wavelet Radiative Transport and Surface Interaction

In Section 3.8.2, we discussed how wavelets show promise for the compact representation of radiance needed by Lucifer. In this chapter, we will see that apart from compression, representing radiance in terms of a wavelet basis with direction expressed in "Nusselt coordinates" makes several calculations of relevance to illumination computation easier.

In particular, we will show how to construct discrete representations of the radiative transport operator $\mathcal{T}$ (defined in Section 2.4.1) and the surface interaction operator $\mathcal{S}$ (defined in Section 2.4.3) in terms of inner products of smoothing functions. The blocked propagation operator $\mathcal{B}$ (defined in Section 2.4.2) can be constructed from the realization of $\mathcal{T}$ and will be presented in Chapter 7. Notice that these all act directly on wavelet coefficients themselves and do not require a full inverse wavelet transform.

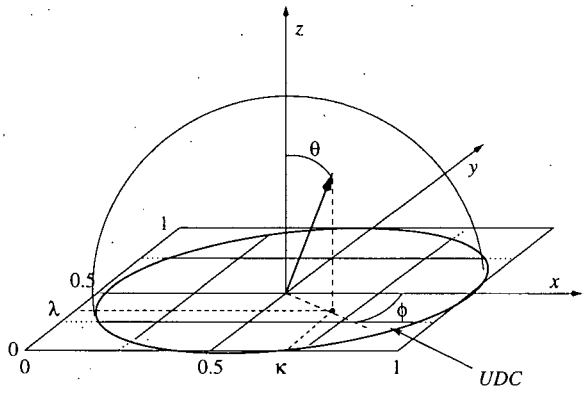

Figure 5.1: Nusselt Coordinates

5.1 Radiance in Nusselt Coordinates

As we discussed in Section 2.4.1, radiance at a point $\mathbf{P}$ in a direction $\mathbf{S}$ is usually represented by a function $L(\mathbf{P}, \mathbf{S})$. If we confine our discussion to surfaces, we can assume a planar (possibly local) parameterization for $\mathbf{P}$ of (u, v). $\mathbf{S}$ is then typically represented in polar and azimuthal coordinates (θ, ϕ) according to the local frame of reference.

Consider the x, y, and z direction cosines corresponding to a direction (θ, ϕ):

$$\mu_x = \sin\theta \cos\phi \quad \mu_y = \sin\theta \sin\phi \quad \mu_z = \cos\theta \tag{5.1}$$

We take (μ_x, μ_y) to be an alternative parameterization of direction.

It is convenient in what follows for all variables to vary between the extrema of 0 and 1, so let us make a change of the directional variables from (μ_x, μ_y), which we used in Section 3.7 to define the UDC, to (κ, λ):

$$\kappa = \frac{\mu_x + 1}{2} \qquad \lambda = \frac{\mu_y + 1}{2} \tag{5.2}$$

Figure 5.1 shows the relation between (θ, ϕ) and (κ, λ) graphically. It also shows

the UDC.

To convert integration over (θ, ϕ) to integration over (κ, λ), the determinant of the Jacobian is:

$$\left|\frac{\partial(\theta,\phi)}{\partial(\kappa,\lambda)}\right| = \frac{4}{\cos\theta \sin\theta} \tag{5.3}$$

so, assuming L_i is zero for directions outside the directional limiting circle, Equation (2.11) becomes

$$L = L_e + 4\int_0^1 \int_0^1 f_r(\mathbf{S}^+, \mathbf{V})\, L_i(\mathbf{S}^+) + f_t(\mathbf{S}^-, \mathbf{V})\, L_i(\mathbf{S}^-)\, d\kappa_i\, d\lambda_i \tag{5.4}$$

That the integral no longer contains trigonometric functions should come as no surprise. We have simply used a differential form of the "Nusselt analog" ([56], but see Cohen and Wallace [14] for a description in English): the amount of power per unit area transferred from a differential solid angle $d\omega_i$ is proportional to $d\kappa_i\, d\lambda_i$, the area of the surface that the projection of $d\omega_i$ on a unit sphere subtends. For this reason, we refer to κ and λ as "Nusselt coordinates".

We also note that, since $\mu_x^2 + \mu_y^2 + \mu_z^2 = 1$ and since each vector is defined only over a hemisphere, not the whole directional sphere, we can express $\mathbf{S}^+$, $\mathbf{S}^-$, and $\mathbf{V}$ all unambiguously in terms of their respective incident and reflected κ's and λ's. Simply put, it is always clear which sign to attach to the square root.

Other ways to parameterize the directional component of a radiance distribution are possible. Light fields (as in Levoy and Hanrahan [44]) and lumigraphs (as in Gortler, et al. [27]), are very promising approaches for display purposes. Christensen, et al. [12] use a combination of a gnomonic projection and "stretch" to map directions to the unit square. None of these approaches, however, leads to the simplification of surface interaction that Equation (5.4) demonstrates.

5.2 Radiance Representation

What are the characteristics of a four-dimensional radiance distribution $L(x, y, \kappa, \lambda)$? The easiest way to visualize this is as "light through a window" where an observer at position (x, y) on a window casts a ray in direction (κ, λ). For a fixed direction, the resulting two-dimensional projection is a parallel projection[1], as shown in the individual cells of Figure 3.5. For a fixed position, the distribution in (κ, λ) would be a "fisheye" view, as shown in the individual cells of Figure 3.6.

In both cases, the result is an image, so we can deal with those radiance distributions as we deal with images.

Radiance at a point on a surface is a potentially discontinuous, generally non-analytic function. We can approximate it with a finite element expansion with N_f degrees of freedom:

$$L(x, y, \kappa, \lambda) = \sum_{j=1}^{N_f} b_j B_j(x, y, \kappa, \lambda) \tag{5.5}$$

Choices for the basis functions B_i include box discretization (*a la FIAT*), Fourier, discrete cosine, spherical harmonics, orthogonal polynomials, and wavelets. We are particularly interested in wavelets because, unlike the other bases listed, their basis functions are of limited support and they can represent discontinuities compactly. They are also capable of considerable compression.

5.3 Multidimensional Wavelets

Appendix A summarizes the properties of one-dimensional wavelets. In this section, we will describe multidimensional wavelets with the intention of applying them to radiative transport and surface interaction.

[1]The special case $(\kappa, \lambda) = (\frac{1}{2}, \frac{1}{2})$ is an orthographic projection.

For D-dimensional coordinates

$$\mathbf{q} = (q_1, q_2, \ldots, q_D), \tag{5.6}$$

we can define a set of multidimensional wavelet basis functions indexed by a *standard multiresolution index*

$$\mathbf{j} = (\nu^j, l_1^j, m_1^j, l_2^j, m_2^j, \ldots, l_D^j, m_D^j) \tag{5.7}$$

where ν^j, which we call the "basis selector", determines the combination of one dimensional smoothing and wavelet functions:

$$B_{\mathbf{j}}(\mathbf{q}) = \begin{cases} \phi_{l_1^j m_1^j}(q_1)\phi_{l_2^j m_2^j}(q_2)\ldots\phi_{l_D^j m_D^j}(q_D) & \nu^j = 0 \\ \psi_{l_1^j m_1^j}(q_1)\phi_{l_2^j m_2^j}(q_2)\ldots\phi_{l_D^j m_D^j}(q_D) & \nu^j = 1 \\ \phi_{l_1^j m_1^j}(q_1)\psi_{l_2^j m_2^j}(q_2)\ldots\phi_{l_D^j m_D^j}(q_D) & \nu^j = 2 \\ \qquad\qquad\vdots & \\ \psi_{l_1^j m_1^j}(q_1)\psi_{l_2^j m_2^j}(q_2)\ldots\psi_{l_D^j m_D^j}(q_D) & \nu^j = 2^D - 1 \end{cases} \tag{5.8}$$

We refer to the special case of $\nu^j = 0$ as the "pure smoothing" component, as the corresponding basis function is made up of only smoothing functions.

We can apply Equation (A.10) multi-dimensionally. If we have $\mathbf{j}$ as in Equation (5.7) and define $\mathbf{k}$ as

$$\mathbf{k} = (\nu^k, l_1^k, m_1^k, l_2^k, m_2^k, \ldots, l_D^k, m_D^k) \tag{5.9}$$

then

$$\left\langle \tilde{B}_{\mathbf{j}} \mid B_{\mathbf{k}} \right\rangle_{\mathbf{q}} = \delta_{\nu^j \nu^k} \prod_{d=1}^{D} \delta_{l_d^j l_d^k} \delta_{m_d^j m_d^k} \tag{5.10}$$

(Note the use of the subscripted inner product described in Section A.3.) $\tilde{B}_{\mathbf{j}}$ is identical to $B_{\mathbf{j}}$ as defined in Equation (5.8), but with the primal wavelets and smoothing functions replaced by their duals.

This is the "standard" multidimensional Cartesian product basis. It is also possible to constrain $l_1^j = l_2^j = \ldots = l_D^j \equiv l^j$, resulting in the so-called "nonstandard" basis. In general multidimensional (especially image-oriented) applications, as cited in Daubechies [16] and in Schröder et al. [65], the nonstandard bases are preferred because of their "square" support.

In this paper, we are considering 4-dimensional, nonstandard basis functions, so let us enumerate the coordinates with the 4-vector

$$\mathbf{q} = (u, v, \kappa, \lambda) \tag{5.11}$$

and the basis functions with a *nonstandard multiresolution index*

$$\mathbf{j} = (\nu, l, m_u, m_v, m_\kappa, m_\lambda). \tag{5.12}$$

Using Equations (A.1), (A.3), (A.5), (A.6) and (5.8), all the basis functions at level l of the pyramid can be written in terms of the $\nu = 0$ basis functions at level $l+1$:

$$B_{\nu l \mathbf{m}}(\mathbf{q}) = 4 \begin{cases} \sum_{\mathbf{m}'} h_{m'_u} h_{m'_v} h_{m'_\kappa} h_{m'_\lambda} B_{0(l+1)(2\mathbf{m}+\mathbf{m}')}(\mathbf{q}) & \nu = 0 \\ \sum_{\mathbf{m}'} g_{m'_u} h_{m'_v} h_{m'_\kappa} h_{m'_\lambda} B_{0(l+1)(2\mathbf{m}+\mathbf{m}')}(\mathbf{q}) & \nu = 1 \\ \sum_{\mathbf{m}'} h_{m'_u} g_{m'_v} h_{m'_\kappa} h_{m'_\lambda} B_{0(l+1)(2\mathbf{m}+\mathbf{m}')}(\mathbf{q}) & \nu = 2 \\ \qquad\qquad \vdots & \\ \sum_{\mathbf{m}'} g_{m'_u} g_{m'_v} g_{m'_\kappa} g_{m'_\lambda} B_{0(l+1)(2\mathbf{m}+\mathbf{m}')}(\mathbf{q}) & \nu = 15 \end{cases} \tag{5.13}$$

where $\mathbf{m}' \equiv (m'_u, m'_v, m'_\kappa, m'_\lambda)$.

So for any function f,

$$\langle f \mid B_{\nu l \mathbf{m}} \rangle_q = \sum_{\mathbf{m}'} W_{\nu \mathbf{m}'} \left\langle f \mid B_{0(l+1)(2\mathbf{m}+\mathbf{m}')} \right\rangle_q \tag{5.14}$$

where $W_{\nu \mathbf{m}'}$ is (4 times) a product of smoothing and wavelet coefficients.

5.4 Irradiance

Irradiance is computed as

$$E(x,y) = \int_{\Omega_{\mathbf{N}}^R} L_i(x,y,\theta,\phi)\ |\mathbf{N}\cdot\mathbf{S}|\ d\omega_i. \tag{5.15}$$

Again making use of Equation (5.3), we have

$$E(x,y) = 4\int_0^1\int_0^1 L_i(x,y,\kappa,\lambda)\, d\kappa_i\, d\lambda_i. \tag{5.16}$$

The limits of the both integrations are 0 and 1, but if L_i is zero outside the UDC (see Figure 5.1), we can safely extend the integration limits to $-\infty$ and $+\infty$.

This allows us to say that if

$$L_i(x,y,\kappa,\lambda) = \sum_{\mathbf{j}} b_{\mathbf{j}} B_{\mathbf{j}}(x,y,\kappa,\lambda) \tag{5.17}$$

then

$$E(x,y) = 4\sum_{\mathbf{j}} b_{\mathbf{j}} \langle B_{\mathbf{j}} \mid 1\rangle_{\kappa,\lambda} \tag{5.18}$$

is the wavelet representation of the irradiance.

The inner products on the right hand side are usually easy to compute in tabular form, if not analytically, making particular use of Equation (A.4) to eliminate many coefficients.

5.5 Power Flux

The power flux passing through an area A is defined as

$$\Phi = \int_A \int_{\Omega_{\mathbf{N}}^R} L_i(x,y,\theta,\phi)\ |\mathbf{N}\cdot\mathbf{S}|\ d\omega_i dA = \int_A E(x,y)\, dA \tag{5.19}$$

If we have a spatial parameterization $(u,v) \leftrightarrow (x,y)$ that maps the unit square to A and back, then we have

$$\Phi = \int_0^1\int_0^1 E(x(u,v),y(u,v))\left|\frac{\partial(x,y)}{\partial(u,v)}\right| du dv. \tag{5.20}$$

If, as above, we take $E(x,y) = 0$ for (x,y) outside of A and if we extend Equation (5.17) to include this parameterization:

$$L_i(x,y,\kappa,\lambda) = \sum_{\mathbf{j}} b_{\mathbf{j}} B_{\mathbf{j}}(u(x,y), v(x,y), \kappa, \lambda), \tag{5.21}$$

then we can incorporate Equation (5.18) to get the wavelet representation of flux:

$$\Phi = 4 \sum_{\mathbf{j}} b_{\mathbf{j}} \left\langle B_{\mathbf{j}} \mid \left| \frac{\partial(x,y)}{\partial(u,v)} \right| \right\rangle_q . \tag{5.22}$$

5.6 Transport

We represent radiance as

$$L(\mathbf{q}) = \sum_{\mathbf{k}} b_{\mathbf{k}} B_{\mathbf{k}}(\mathbf{q}) \tag{5.23}$$

where

$$b_{\mathbf{k}} = \left\langle L \mid \tilde{B}_{\mathbf{k}} \right\rangle_{\mathbf{q}} \tag{5.24}$$

and $\mathbf{k}$ is defined as in Equation (5.9).

Radiance travels from a source point $\mathbf{q}_s$ to a destination point $\mathbf{q}_d$. If we have a mapping of $\mathbf{q}_s \rightarrow \mathbf{q}_d$, we can compute

$$L_d(\mathbf{q}_d) = \sum_{\mathbf{k}} b^d_{\mathbf{k}} B_{\mathbf{k}}(\mathbf{q}_d) \tag{5.25}$$

where

$$\begin{aligned} b^d_{\mathbf{k}} &= \left\langle L_s(\mathbf{q}_s(\cdot)) \mid \tilde{B}_{\mathbf{k}} \right\rangle_{\mathbf{q}_d} \\ &= \sum_{\mathbf{j}} b^s_{\mathbf{j}} T_{\mathbf{jk}}, \end{aligned} \tag{5.26}$$

$\mathbf{j}$ is defined as in Equation (5.7), and we define geometry-dependent "transport coefficients"

$$T_{\mathbf{jk}} \equiv \left\langle B_{\mathbf{j}}(q_s(\cdot)) \mid \tilde{B}_{\mathbf{k}} \right\rangle_{\mathbf{q}_d} . \tag{5.27}$$

Using the multidimensional refinement shown in Equation (5.13), given $T_{(0l_j\mathbf{m}_j)\mathbf{k}}$ on level l_j, we can compute all coefficients on the coarser level above it in the pyramid:

$$T_{(\nu_j(l_j-1)\mathbf{m}_j)\mathbf{k}} = \sum_{\mathbf{m}'} W_{\nu_j\mathbf{m}'} T_{0l_j(2\mathbf{m}_j+\mathbf{m}')\mathbf{k}} \tag{5.28}$$

and given $T_{\mathbf{j}(0l_k\mathbf{m}_k)}$ on level l_k, we can compute

$$T_{\mathbf{j}(\nu_k(l_k-1)\mathbf{m}_k)} = \sum_{\mathbf{m}''} W_{\nu_k\mathbf{m}''} T_{\mathbf{j}(0l_k(2\mathbf{m}_k+\mathbf{m}''))} \tag{5.29}$$

where $\mathbf{m}_j = (m_u^j, m_v^j, m_\kappa^j, m_\lambda^j)$ and $\mathbf{m}_k = (m_u^k, m_v^k, m_\kappa^k, m_\lambda^k)$. This means that we can compute all transport coefficients strictly in terms of pure smoothing components:

$$T_{\mathbf{jk}} = \sum_{\mathbf{m}'}\sum_{\mathbf{m}''} W_{\nu_j\mathbf{m}'} W_{\nu_k\mathbf{m}''} T_{(0l_j(2\mathbf{m}_j+\mathbf{m}')(0l_k(2\mathbf{m}_k+\mathbf{m}'')} \tag{5.30}$$

5.7 Surface Interaction

Using Equation (5.8), let us define a mixed primal-dual, four-dimensional, nonstandard wavelet basis:

$$F_{\mathbf{j}}(\kappa_s,\lambda_s,\kappa_r,\lambda_r) = \begin{cases} \tilde{\phi}_{lj m_\kappa^i}(\kappa_s)\tilde{\phi}_{lj m_\lambda^i}(\lambda_s)\phi_{lj m_\kappa^j}(\kappa_r)\phi_{lj m_\lambda^j}(\lambda_r) & \nu^j = 0 \\ \tilde{\psi}_{lj m_\kappa^i}(\kappa_s)\tilde{\phi}_{lj m_\lambda^i}(\lambda_s)\phi_{lj m_\kappa^j}(\kappa_r)\phi_{lj m_\lambda^j}(\lambda_r) & \nu^j = 1 \\ \tilde{\phi}_{lj m_\kappa^i}(\kappa_s)\tilde{\psi}_{lj m_\lambda^i}(\lambda_s)\phi_{lj m_\kappa^j}(\kappa_r)\phi_{lj m_\lambda^j}(\lambda_r) & \nu^j = 2 \\ \vdots & \\ \tilde{\psi}_{lj m_\kappa^i}(\kappa_s)\tilde{\psi}_{lj m_\lambda^i}(\lambda_s)\psi_{lj m_\kappa^j}(\kappa_r)\psi_{lj m_\lambda^j}(\lambda_r) & \nu^j = 15 \end{cases} \tag{5.31}$$

where

$$\mathbf{j} = (\nu_j, l_j, m_\kappa^i, m_\lambda^i, m_\kappa^j, m_\lambda^j) \tag{5.32}$$

So $F_{\mathbf{j}}(\kappa_s,\lambda_s,\kappa_r,\lambda_r)$ is $B_{\mathbf{j}}(\kappa_s,\lambda_s,\kappa_r,\lambda_r)$ with dual scaling functions and wavelets substituted for the primal scaling functions and wavelets in the incident directional components only.

If we then represent the BRDF in Nusselt coordinates with this basis:

$$f_r(\kappa_s, \lambda_s, \kappa_r, \lambda_r) = \sum_{\mathbf{j}} f_{\mathbf{j}}^r F_{\mathbf{j}}(\kappa_s, \lambda_s, \kappa_r, \lambda_r) \tag{5.33}$$

and

$$L_i(x, y, \kappa_s, \lambda_s) = \sum_{\mathbf{k}} b_{\mathbf{k}} B_{\mathbf{k}}(x, y, \kappa_s, \lambda_s) \tag{5.34}$$

where

$$\mathbf{k} = (\nu_k, l_k, m_u^k, m_v^k, m_\kappa^k, m_\lambda^k) \tag{5.35}$$

then, applying Equation (5.4) and again requiring either f_r or L_i (or both) to vanish outside the UDC (thus allowing us to extend our integration to $(-\infty..\infty)$), the reflected radiance is

$$\begin{aligned} L_r &= 4\int_0^1 \int_0^1 f_r(\mathbf{S}^+, \mathbf{V})\, L_i(\mathbf{S}^+)\, d\kappa_s\, d\lambda_s \\ &= 4\sum_{\mathbf{j}}\sum_{\mathbf{k}} f_{\mathbf{j}}^r b_{\mathbf{k}} \langle F_{\mathbf{j}} \mid B_{\mathbf{k}} \rangle_{\kappa_s, \lambda_s} \end{aligned} \tag{5.36}$$

We seek a wavelet representation of the post-interaction radiance:

$$L_r = \sum_{\mathbf{n}} b_{\mathbf{n}}^r B_{\mathbf{n}}(x, y, \kappa_r, \lambda_r) \tag{5.37}$$

Again using the basis representation of the reflected radiance as in Equations 5.23 and 5.24, it follows that

$$b_{\mathbf{n}}^r = \left\langle L_r \mid \tilde{B}_{\mathbf{n}} \right\rangle_{x, y, \kappa_r, \lambda_r} \tag{5.38}$$

So, substituting Equation 5.36, we find

$$b_{\mathbf{n}}^r = 4\sum_{\mathbf{j}}\sum_{\mathbf{k}} f_{\mathbf{j}}^r b_{\mathbf{k}} \left\langle \langle F_{\mathbf{j}} \mid B_{\mathbf{k}} \rangle_{\kappa_s, \lambda_s} \mid \tilde{B}_{\mathbf{n}} \right\rangle_{x, y, \kappa_r, \lambda_r} \tag{5.39}$$

Let us now simplify notation. We can rewrite $F_{\mathbf{j}}$ and $B_{\mathbf{k}}$ compactly by defining a function $\xi_{lm}^{\beta}(x)$ that takes on the value of the smoothing function or the

wavelet depending on a single binary value β:

$$\xi^{\beta}_{lm}(x) = \begin{cases} \phi_{lm}(x) & \beta = 0 \\ \psi_{lm}(x) & \beta = 1 \end{cases} \tag{5.40}$$

and similarly for a $\tilde{\xi}$ with $\tilde{\phi}$ and $\tilde{\psi}$ in place of ϕ and ψ, respectively.

We also take indexable (from 0 to 3) representations of the arguments to $F_{\mathbf{j}}$ and $B_{\mathbf{k}}$, respectively

$$\mathbf{p} = (\kappa_s, \lambda_s, \kappa_r, \lambda_r) \tag{5.41}$$

and

$$\mathbf{q} = (u, v, \kappa_s, \lambda_s). \tag{5.42}$$

If we now adopt the notation where $i_{\langle\alpha\rangle}$ refers to the αth bit of the binary form of i (i.e. $i2^{-\alpha} \bmod 2$), we can express the innermost inner product of Equation 5.39 as

$$\langle F_{\mathbf{j}} \mid B_{\mathbf{k}} \rangle_{\kappa_s,\lambda_s} = \int_0^1 \int_0^1 \left[\prod_{\alpha=0}^{1} \tilde{\xi}^{\nu^j_{\langle\alpha\rangle}}_{lj m^j_\alpha}(p_\alpha) \xi^{\nu^k_{\langle\alpha\rangle}}_{lk m^k_\alpha}(q_\alpha) \prod_{\alpha=2}^{3} \xi^{\nu^j_{\langle\alpha\rangle}}_{lj m^j_\alpha}(p_\alpha) \xi^{\nu^k_{\langle\alpha\rangle}}_{lk m^k_\alpha}(q_\alpha) \right] d\kappa_s \, d\lambda_s \tag{5.43}$$

and removing factors that do not depend on the variables of integration from the integral, we have

$$\begin{aligned} \langle F_{\mathbf{j}} \mid B_{\mathbf{k}} \rangle_{\kappa_s,\lambda_s} \quad = \quad & \left\langle \tilde{\xi}^{\nu^j_{\langle 0\rangle}}_{lj m^j_0} \mid \xi^{\nu^k_{\langle 2\rangle}}_{lk m^k_2} \right\rangle_{\kappa_s} \left\langle \tilde{\xi}^{\nu^j_{\langle 1\rangle}}_{lj m^j_1} \mid \xi^{\nu^k_{\langle 3\rangle}}_{lk m^k_3} \right\rangle_{\lambda_s} \\ & \xi^{\nu^k_{\langle 0\rangle}}_{lk m^k_0}(x) \, \xi^{\nu^k_{\langle 1\rangle}}_{lk m^k_1}(y) \, \xi^{\nu^j_{\langle 2\rangle}}_{lj m^j_2}(\kappa_r) \, \xi^{\nu^j_{\langle 3\rangle}}_{lj m^j_3}(\lambda_r) \end{aligned} \tag{5.44}$$

If we now define a *multiresolution delta tensor*

$$\Delta^{\sigma\tau}_{\mathbf{jk}} = \left\langle \tilde{\xi}^{\nu^j_{\langle\sigma\rangle}}_{lj m^j_\sigma} \mid \xi^{\nu^k_{\langle\tau\rangle}}_{lk m^k_\tau} \right\rangle \tag{5.45}$$

we can rewrite Equation 5.44 as

$$\langle F_{\mathbf{j}} \mid B_{\mathbf{k}} \rangle_{\kappa_s,\lambda_s} = \Delta^{02}_{\mathbf{jk}} \, \Delta^{13}_{\mathbf{jk}} \, \xi^{\nu^k_{\langle 0\rangle}}_{lk m^k_0}(x) \, \xi^{\nu^k_{\langle 1\rangle}}_{lk m^k_1}(y) \, \xi^{\nu^j_{\langle 2\rangle}}_{lj m^j_2}(\kappa_r) \, \xi^{\nu^j_{\langle 3\rangle}}_{lj m^j_3}(\lambda_r) \tag{5.46}$$

and we can continue to apply the Δ symbol to simplify the whole nested inner product in Equation 5.39

$$\left\langle \langle F_{\mathbf{j}} \mid B_{\mathbf{k}} \rangle_{\kappa_s,\lambda_s} \mid \tilde{B}_{\mathbf{n}} \right\rangle_{x,y,\kappa_r,\lambda_r} = \Delta^{02}_{\mathbf{jk}} \Delta^{13}_{\mathbf{jk}} \Delta^{00}_{\mathbf{kn}} \Delta^{11}_{\mathbf{kn}} \Delta^{22}_{\mathbf{jn}} \Delta^{33}_{\mathbf{jn}}. \tag{5.47}$$

The Δs are very much dependent upon our choice of wavelet, so we will defer discussing their computation to Chapter 7.

We can do the same thing with a BTDF and so represent general surface interactions: reflection, refraction, and transmission.

Chapter 6

Wavelet Radiative Transfer Implementation and Practicum

In this chapter, we discuss the implementation of the WRT algorithm (hereafter, "*WRT*"), along with a practicum ("a course of study ... that involves the supervised practical application ... of previously studied theory"). We apply some of the concepts of Chapter 5 to a classic illumination problem: the transport of radiation between two arbitrarily-oriented polygons. Before that, however, we describe naming conventions, design decisions, and classes that apply to both *WRT* and the *Lucifer* implementation we discuss in Chapter 7.

These implementations take up approximately 30,000 lines of C code. We therefore emphasize that the classes and pseudocode we present here are in most cases simplified for clarity.

Class	Prefix	Meaning
Polygon	pgn	a 2D polygon with an arbitrary number of edges
Transform	tf	a conventional 4×4 affine 3D transform
Transform2d	tf2d	a conventional 3×3 affine 2D transform
TransportGeometry	tg	geometry for WRT
WaveletCoefficientTree	wct	a WCT (sparse tree of wavelet coefficients)
WaveletIndex	wi	indexes a wavelet coefficient
WaveletNode	wn	a single node of a WCT (16 `float` values per channel)

Table 6.1: Major *WRT* Classes and Their Abbreviations.

6.1 Naming Conventions

Throughout *WRT* and *Lucifer* implementations, we have followed certain programming conventions that we feel have improved reliability and flexibility and allowed many of the benefits of object-oriented programming while developing in a highly portable environment[1]. We describe those conventions here to enhance the readability of what follows.

The most visible conventions are those we use to name objects (i.e., C data structures) and their associated methods. They are an adapted form of what are known as "Hungarian" conventions[2], partly in deference to the nationality of their chief developer, Charles Simonyi [70].

As implemented here, all classes have C `typedefs` associated with them. Their names are mixed case. For example, three dimensional vectors have a `typedef` "Vector". Each class also has a lower-case prefix. Every instance of that class (i.e., object) has a name that begins with that abbreviation. The rest of the object name describes the particular object in mixed case. An example of the "Vector"

[1] Indeed, the code moves between IBM AIX, SGI IRIX, and Intel (RedHat) Linux with fewer than 50 system-dependent lines of code, mostly due to differing system header files.

[2] See McConnell [50] for an additional discussion of Hungarian naming, particularly as practiced at Microsoft.

$l_{\max}$	1	2	3	4	5	6	7	8
Number of Bits	9	14	18	23	27	31	35	40
$l_{\max}$	9	10	11	12	13	14	15	16
Number of Bits	44	48	52	56	60	64	68	73

Table 6.2: Number of Bits Required to Store a Nonstandard Multiresolution Index as a Function of the Maximum Resolution Level

class might be "vNormal". The prefix, followed by a "_", also prefaces the names of ("member") functions that take a data structure (or a pointer to such a data structure) as a first parameter. This allows us to represent the object-oriented concept of a "method" in C. For instance, we have a function *v_mag()* that returns the magnitude of its first (and only) argument, a *Vector*. Table 6.1 lists the major classes of *WRT* (and *Lucifer*) with their prefixes.

6.2 Indexing Wavelets

Wavelet indices contain the six integer values referred to in Section 5.1:

- the basis selector $\nu \in \{0 \ldots 15\}$, as used in Equation (5.8),
- the level $l \in \{0 \ldots l_{\max}\}$, where $l_{\max}$ is the maximum depth of resolution, and
- a vector of four offset values $\mathbf{m}$ with each component $m_i \in \left\{0 \ldots 2^l - 1\right\}$ (for Haar).

Theoretically, the total number of bits required to represent a wavelet index is $4 + \lceil \log_2(l_{\max} + 1) \rceil + 4l_{\max}$. Table 6.2 shows this quantity for several possible values of $l_{\max}$. If we represented the coefficients as single channel 32-bit floating

```
typedef struct {
    short int nu;
    short int l;
    short int m[4];
} WaveletIndex;
```

Figure 6.1: The *WaveletIndex* `typedef`

point values, then for any value of $l_{max} > 6$ we would use more storage for indices than for coefficients.

For speed of access, it is advantageous to use data that is at least 8-bit byte-aligned.

6.3 The *WaveletIndex* Class

Figure 6.1 shows the wavelet index `typedef` we have adopted for our implementation. The names and sequence of the components are consistent with Equation 5.12.

For the time being, we have implemented all of these fields as `short` (16 bit) unsigned integers. This allows us to go as far as level $wi.l = 16$ without overflowing the offsets. Experience suggests that the maximum level we will use will be much less than 16. We could achieve a minor reduction in memory usage by using `unsigned char` variables for $wi.nu$[3].

The implemented structure requires 96 bits, which is 19 bits (31 %) larger than the theoretical minimum given in Table 6.2, but as we shall see in the next section, it is possible to group coefficients in such a way that the overhead of having to store their indices is minimal.

[3]but not $wi.l$, since it needs to represent levels from 0 to the maximum level inclusively.

6.4 Storing Wavelet Coefficients

Because wavelet coefficients are hierarchical in nature, we refer to the data structures we devise here to store them as "wavelet coefficient trees" (hereafter, "WCT"s).

Representing a WCT with a maximum level of resolution $l_{\max}$ requires $16^{l_{\max}+1}$ possible wavelet coefficients per channel. Clearly, compression is called for. In Section A.7, we describe how wavelet properties provide an L^2-optimal compression strategy – thresholding low-magnitude coefficients. In this case, the data was single-channel and single dimensional (i.e., each wavelet coefficient's support was unique), but the results are independent of the wavelet dimensionality.

We not only want to guarantee a good approximation of the data with the compressed coefficients, but also efficient use of storage and fast reconstruction. We will give an example of wavelet compression in Section 6.13.

6.4.1 Hashing Coefficients

Given a sparse set of wavelet radiance coefficients $\{b_{\mathbf{k}}\}$, we need to store them in a way that facilitates the mapping of the wavelet index $\mathbf{k} \rightarrow b_{\mathbf{k}}$ needed to perform the transport operation Equation (5.26). The obvious way to do this is with a hash table. Not knowing the set of destination indices $\{\mathbf{k}\}$ in advance prohibits perfect hashing, so it is necessary that the hashing scheme allows for collisions and that the hash table entries contain $\mathbf{k}$ as well as $b_{\mathbf{k}}$ values.

6.4.2 Multichannel Grouping

As we mentioned in Section 2.4.1, we have been treating data monochromatically throughout this thesis. In practical applications, however, we must evaluate Equation (5.26) for all three (or however many) channels. Having assumed a non-

participating medium, the transport coefficients $T_{\mathbf{jk}}$ are achromatic. Evaluation of Equation (5.26) simply means multiplying each element of a (now) 3-vector $b^s_{\mathbf{j}}$ by the same value of $T_{\mathbf{jk}}$ and accumulating the result in another 3-vector $b^d_{\mathbf{k}}$.

In the absence of the need for compression, it would be convenient to group all channels of $b^s_{\mathbf{j}}$ into a single group, rather than create a separate representation for each channel. In the presence of compression, however, each channel has its own threshold. If one component is above its threshold but the other two are below theirs, saving the latter is a waste of storage. This would be mitigated, however, if there were a high degree of correlation between the magnitudes of the coefficients. Certainly, a wavelet representation of white light given off by a luminaire displays a high degree of correlation. When this light is reflected off a surface of varying spectral reflectivity, however, that correlation will be diminished. By how much depends on the nature of the surface.

We have two choices here: to group or not to group multichannel data. For the time being, we have chosen the former – believing that there is sufficient correlation and advantage in retrieval speed in most situations to justify grouping. This definitely requires further study.

6.4.3 Hashing Nodes Instead of Coefficients

In one dimension, a node in the wavelet pyramid contains a single wavelet coefficient. In four dimensions, such a node has one pure wavelet ($\nu = 15$) coefficient and fourteen mixed wavelet/smoothing ($\nu \in \{1 \ldots 14\}$) coefficients. (Recall that the pure smoothing ($\nu = 0$) coefficient may be reconstructed from the node's ancestors, if any, and only needs to be kept at the root node.) All coefficients at a given node correspond to basis functions with the same (Haar) or similar (other bases) support.

```
typedef struct {
    int iWnOfHash[int mxnIWnOfHash];
    struct WaveletNode {
        WaveletIndex wi;
        unsigned short mskChild;
        unsigned short mskNu;
        int iWclFirst;
    } wnBase[int nWn];
    struct WaveletCoefficientList {
        float b[3];
        int iWclNext;
    } wclBase[int nWcl];
    double umrad;
} WaveletCoefficientTree;
```

Figure 6.2: The *WaveletCoefficientTree* `typedef`

We might therefore expect to find a higher degree of correlation in magnitude between coefficients that belong to the same node and coefficients that do not. This suggests that we can reduce the index storage overhead by hashing entire nodes rather than individual coefficients.

6.5 The *WaveletCoefficientTree* Class

Figure 6.2 shows the fundamental structure we use to represent wavelet coefficients: *WaveletCoefficientTree*. Note that the code in the figure is not legal C code: we have combined several subsidiary classes and moved some component definitions around to indicate dynamically-sized structures in an obvious (we hope) way.

Given a *WaveletCoefficientTree wct*, *wct.iWnOfHash[]* is the hash table itself. It is indexed by the hash of a (pure smoothing) wavelet index. Its entries are indices into *wct.wnBase[]*, the array of wavelet nodes. Note that the use of integer indices rather than pointers allows the dynamic resizing of *wct.wnBase[]*.

Each node contains the corresponding wavelet index. This is not only use-

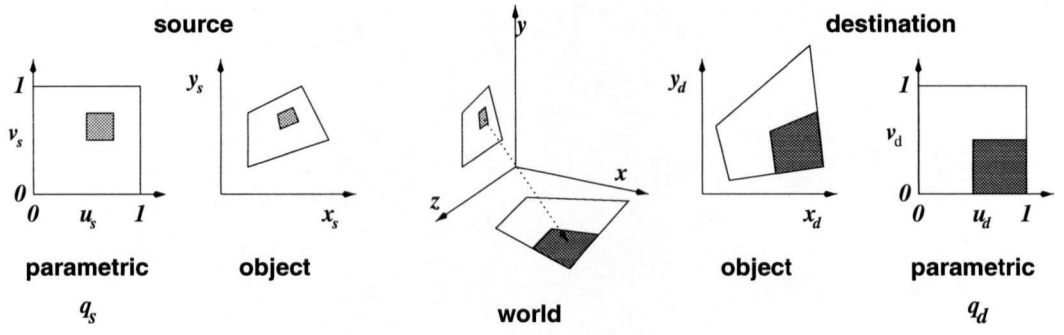

Figure 6.3: Coordinate Systems. The gray quadrilaterals represent the support of source and destination bases.

ful for hash collision detection, but by indexing *wct.wnBase[]*, we can traverse all nonzero entries in *wct* directly. *wct.wnBase[i].mskChild* is a 16-bit mask indicating which of node i's 16 children are also present in *wct*. *wct.wnBase[i].mskNu* is a 16-bit mask indicating which basis selectors are present in *wct* for node i.

wct.wnBase[i].iWclFirst is the index of the first element of a list of wavelet coefficients in *wct.wnBase[i].wclBase[]*. The elements are ordered in increasing basis selector value, which is derived from *wct.wnBase[i].mskNu*. Each element *wct.wnBase[i].wclBase[j]* contains the index *wct.wnBase[i].wclBase[j].iWclNext* of the next-higher set of multichannel coefficients belonging to node i. (The last element in the list has this index set to -1.)

6.6 Representing Transport Geometry

To establish the $\mathbf{q}_s \leftrightarrow \mathbf{q}_d$ mapping we need in order to transport radiance from source to destination, we must deal with several coordinate systems, as shown in Figure 6.3: source parametric, source object, world, destination object, and destination parametric. Obviously, source object to destination object is best done with

a conventional affine transform (with projection), but there are several choices possible for the parametric $\leftrightarrow$ object mappings: rectilinear, perspective, and bilinear. All of these are local to the sending or receiving surface.

6.6.1 Rectilinear

This is the simplest possible mapping:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} W & 0 \\ 0 & H \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \tag{6.1}$$

W and H are the dimensions of a bounding rectangle. If the object is a rectangle, an obvious strategy is to choose coordinates in which W is its width and H is its height. This will ensure that the (inverse) transformed object completely fills the unit square. Otherwise, or in the more general case of an arbitrarily-sided polygon, when computing $T_{\mathbf{jk}}$ we must clip the (square) support of $B_{\mathbf{j}}$ (source) or $B_{\mathbf{k}}$ (destination) against the polygon when integrating.

A major advantage of the rectilinear mapping is the ease of computation of the Jacobian determinant:

$$\left|\frac{\partial(x,y)}{\partial(u,v)}\right| = WH \tag{6.2}$$

which makes the power flux computation shown in Equation (5.22) very easy:

$$\Phi = 4WH \sum_{\mathbf{j}} b_{\mathbf{j}} \left\langle B_{\mathbf{j}} \mid 1 \right\rangle_q . \tag{6.3}$$

especially in the case of Haar wavelets:

$$\Phi = 4WHb_{\mathbf{0}}. \tag{6.4}$$

6.6.2 Perspective

This mapping allows us to represent the more general quadrilaterals without the need to clip:

$$\begin{bmatrix} xw \\ yw \\ w \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{6.5}$$

where the A_{ij}'s are easily-determined functions of the quadrilateral vertices.

A straight line in perspective parametric coordinates $au + bv + c = 0$ transforms to a straight line $a'x + b'y + c'$ in object coordinates. This means that a quadrilateral in object coordinates will map to a quadrilateral in parametric coordinates and vice versa. This has favourable implications for transport coefficient computation that we will discuss below.

One drawback of a perspective mapping is that if the quadrilateral approaches degeneracy (a triangle, for instance), the appearance of a uniform grid in parametric space becomes increasingly nonuniform.

The power flux computation of Equation (5.22) is not as easy as in the rectilinear case, but may still be analytically done for basis functions B_j with closed-form representations, such as splines.

6.6.3 Bilinear

As with 2-D perspective, this mapping also allows us to represent quadrilaterals without clipping. If the quadrilateral is defined by four points $\{\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3\}$ (in CCW order), the customary bilinear mapping applies:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1-v & v \end{bmatrix} \begin{bmatrix} \mathbf{p}_0 & \mathbf{p}_1 \\ \mathbf{p}_3 & \mathbf{p}_2 \end{bmatrix} \begin{bmatrix} 1-u \\ u \end{bmatrix} \tag{6.6}$$

```
typedef struct {
    Transform tfSobjWld;
    Transform tfDobjWld;
    Transform tfSobjDobj;
    Transform tfR;
    Polygon *pgnSobj;
    Transform2d tf2dSparToSobj;
    Transform2d tf2dSobjToSpar;
    Polygon *pgnDobj;
    Transform2d tf2dDparToDobj;                                    10
    Transform2d tf2dDobjToDpar;
} TransportGeometry;
```

Figure 6.4: The *TransportGeometry* `typedef`

Unlike the perspective case, we can treat triangles as degenerate quadrilaterals, albeit with some irregular object space meshing. However, this mapping is not without its own drawbacks. While the parametric-to-object mapping is straightforward, the inverse object-to-parametric mapping is, in fact, double valued: a given (x, y) usually has two solutions (u, v), one of them inside the unit rectangle, one outside. In order to distinguish the two cases, we must clip in object space before inversion.

A more serious drawback is that straight lines are not, in general, preserved. The inverse projection of a straight line $ax + by + c = 0$ into parametric space is, in general, a hyperbola. This also has implications for transport coefficient computation: it complicates determination of the limits of integration.

Again, while the still more complicated Jacobian of this transform makes the power flux computation of Equation (5.22) more difficult, it may still be done for choices of B_j with closed-form representations, such as splines.

6.7 The *TransportGeometry* Class

The *TransportGeometry* `typedef` shown in Figure 6.4 contains all geometric information necessary to compute wavelet radiative transfer between source and destination polygonal objects. Given a *TransportGeometry* object *tg*, *tg.tfSobjWld* is the affine 3D transform from source object to world positional coordinates, *tg.tfWldDobj* is the affine 3D transform from world to destination object positional coordinates, *tg.tfR* is the orthogonal 3D rotational transform from source to destination directional coordinates, *tg.pgnSobj* and *tg.pgnDobj* are the source and destination polygons (in object coordinates), *tg.tf2dSparToSobj* is the affine 2D transform from source parametric to source object coordinates, and *tg.tf2dDobjToDpar* is the affine 2D transform from source parametric to source object coordinates. The implemented *TransportGeometry* class contains inverses of most of these transforms as well, since some of the integration schemes we use actually project destination points back into source space.

6.8 Choice of Wavelet

Except where noted, our discussions in Appendix A and Chapter 5 did not depend on any particular choice of wavelet. For implementation purposes, we have to choose one. There are several good reasons for choosing Haar wavelets.

As we mentioned in Section A.6, the fast wavelet transform can be performed in $O(N)$ time, but if we allow for a varying dimensionality D and wavelet basis, it is easy to see from Equation (5.13) that the complexity is actually $O(W_h^D N)$ where W_h is the (varying) maximum width of the $\{h_j\}$ and $\{\tilde{h}_j\}$ (and, consequently, $\{g_j\}$ and $\{\tilde{g}_j\}$) coefficient sets.

For this reason, as the dimensionality increases, the rapidly-increasing operation count makes narrower filters more and more desirable, even though wider filters generally have better approximation properties. Since Haar is the narrowest possible wavelet filter ($W_h = 2$), it seems a wise strategy to make any multidimensional efforts first with Haar and move to wider bases later if Haar proves unsatisfactory.

An additional advantage of Haar wavelets over the others is the simplification of the calculation of the transport coefficients, irradiance and power flux. As Equations (5.28) and (5.29) have shown, these coefficients can be computed entirely in terms of pure smoothing calculations. A four-dimensional Haar pure smoothing basis is a function that is constant ($= 4^l$) within a hypercube and zero outside of it. The resulting transport coefficients are volume integrals of the overlap between such a hypercube in destination parametric space and the object which is a projection of a hypercube in source parametric space into the destination space[4].

6.9 Problems with Transport Coefficient Computation

As the preceding subsection suggests, using Haar wavelets turns transport coefficient computation into volume integral evaluation. There are two practical problems that complicate the computation of that volume.

6.9.1 Some Source Points Do Not Project Into Destination Space

The source hypercube defines a range of positional and directional coordinates $\mathbf{q}_s$. Not all of these coordinates may map to points in the destination plane, much less the destination quadrilateral. This complicates any attempt at direct evaluation of

[4]This may not be such a great simplification. After all, any arbitrarily complex 3-dimensional integral may be trivially turned into a 4-dimensional volume integral!

the transport integrals.

6.9.2 The Projected Hypercube Has Curved Sides

Even if all points in the source hypercube map to the destination plane, the nature of the resulting volume, is not trivial. Needless to say, the projection of a source parametric hypercube into destination parametric space is not a hypercube. If, however, we could choose coordinate systems such that the source hypercube mapped to a polytope in destination space, we could take advantage of computational geometric techniques to first clip it against the destination hypercube and then compute the volume of the resulting polytope. Unfortunately, this is not possible because the source hypercube does not project to a polytope.

Consider the coordinate systems described in Section 6.6. Even if we choose rectilinear parametric $\leftrightarrow$ object mappings, the source object to destination object transform involves a projection. Hence, at best, the $q_s \rightarrow q_d$ mapping has a nonlinear dependence on the directional components.

As a result, the projection of the source hypercube into destination parametric space has curved sides. Furthermore, the curvature is such that we cannot guarantee that the convex hull of the polytope formed by projecting the 16 corners of the source hypercube into destination space contains the hypervolume[5].

6.10 Integration Techniques

For these reasons, we must resort to multidimensional numerical integration schemes. Before considering candidate techniques, we first make an observation about the di-

[5]We have not fully pursued the possibility of an approximation of the projected hypercube by its convex hull here.

mensionality required for numerical integration.

6.10.1 Reducing the Dimensionality

As Equation (5.27) indicates, the computation of transport coefficients is intrinsically four-dimensional: two directional integrals and two positional integrals. If we take the two outermost integrals over direction and restrict our discussion to pure Haar smoothing components ($\nu_j = \nu_k = 0$, as Equation (5.30) permits), it is then evident that

$$T_{\mathbf{jk}} = 4^{l_j+l_k} \iint_{S_\mathbf{k}} G_\mathbf{j}(\kappa_d, \lambda_d)\, A_\mathbf{jk}(\kappa_d, \lambda_d)\, d\kappa_d\, d\lambda_d \tag{6.7}$$

$S_\mathbf{k}$ is the (square) directional support of $\tilde{B}_\mathbf{k}$. $G_\mathbf{j}(\kappa_d, \lambda_d)$ is a geometric function which is equal to one if a ray from the destination plane projected backwards along the (κ_d, λ_d) direction reaches the source plane and falls within the directional support of $B_\mathbf{j}$. Otherwise, it is zero. $A_\mathbf{jk}(\kappa_d, \lambda_d)$ is the area of the intersection of the spatial support of $B_\mathbf{k}$ in destination parametric space with the projection (in the κ_d, λ_d direction) to destination parametric space of the spatial support of $B_\mathbf{j}$.

We are now in a position to evaluate the coordinate mappings given in Section 6.6 to see which of them makes $A_\mathbf{jk}(\kappa_d, \lambda_d)$ easy to compute. All the mappings transform lines of constant u or v to lines in object space, so any of them would work for the source parametric to source object mapping. Only rectilinear and perspective mappings, however, transform arbitrary lines in object space to lines in parametric space. If we choose either of them for our destination object to destination parametric mappings, computation of $A_\mathbf{jk}(\kappa_d, \lambda_d)$ amounts to clipping the projected quadrilateral to the spatial support of $B_\mathbf{k}$ using a conventional polygon clipping algorithm and computing the resulting area. This allows us to reduce the dimensionality that we need to integrate numerically from four to two.

6.10.2 Numerical Quadrature

Regardless of the number of dimensions, numerical integration techniques are all based on some form of quadrature:

$$\int f(\mathbf{x})d\mathbf{x} \approx \sum_{i=1}^{N_{samp}} w_i f(\mathbf{x}_i) \tag{6.8}$$

where N_{samp} is the number of samples. Techniques differ principally in their choices of weights w_i and sample points $\mathbf{x}_i$. Zwillinger [87] provides an extensive survey of these. The ones we have chosen to evaluate are:

- *trapezoidal*: a regular grid approximating f linearly in each dimension
- *Romberg*: a multilevel (Richardson) extrapolation (in terms of the grid spacing) of trapezoidal results
- *Monte Carlo*: $\mathbf{x}_i$'s chosen from a (possibly stratified) pseudo-random sequence[6]
- *Halton*: similar to Monte Carlo, but using *quasi-random* numbers generated according to number theoretical considerations
- *Hammersley*: an alternative quasi-random method

While pseudo-random and quasi-random techniques are generally preferred for multidimensional quadrature, we include trapezoidal and Romberg techniques to explore the two-dimensional case, where Monte Carlo has error of order $O(N_{\text{samp}}^{-1/2})$[7] and the trapezoidal rule has error of order $O(N_{\text{samp}}^{-3/2})$. We must be careful to use these error estimates cautiously, however, since our integrand, $G(\kappa_d, \lambda_d)\, A_{\mathbf{jk}}(\kappa_d, \lambda_d)$ contains discontinuities that error analyses do not account for.

[6] Glassner [24] is a good overview of pseudo- and quasi-random integration methods in image synthesis.

[7] This is true regardless of the dimensionality of the problem and is one of the appeals of Monte Carlo integration.

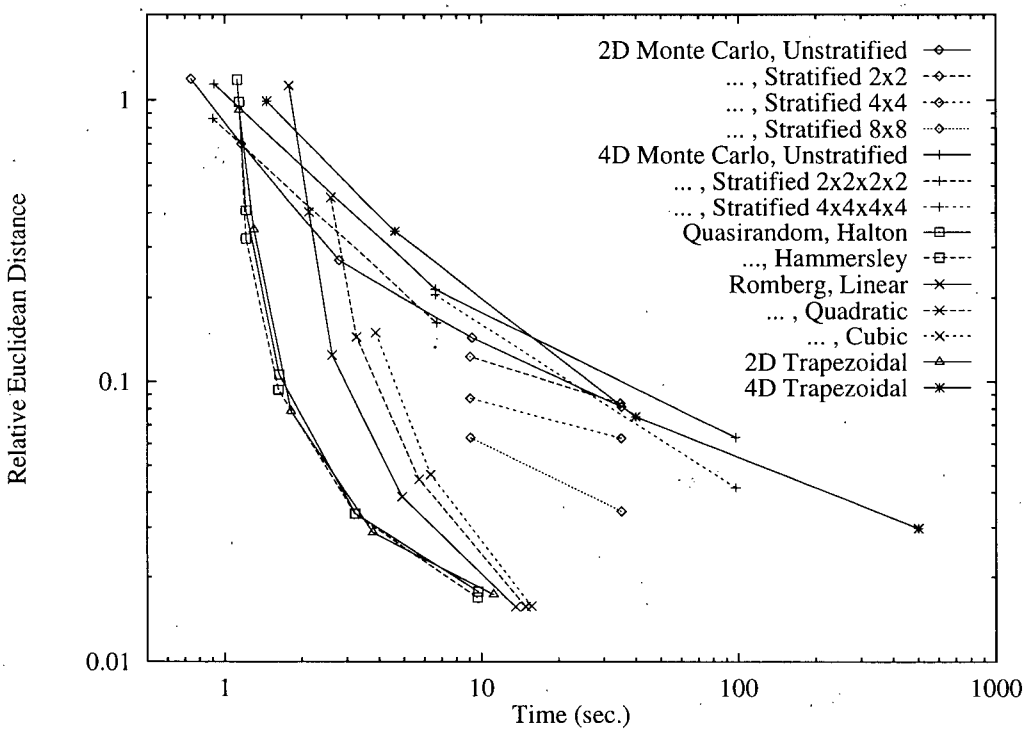

Figure 6.5: Relative Euclidean Distance vs. Time for Different Integration Techniques and Numbers of Samples

6.10.3 Comparison

To evaluate the accuracies of the various methods, we have applied them to a test problem: computing the set of all $T_{\mathbf{jk}}$ for a given $\mathbf{j}$ and geometry. To analyze the results we can treat this set as a vector in a K-dimensional space, where $K = 16^{l_{\max}+1}$ and $l_{\max}$ is the maximum level we have coefficients for. (I.e., we have K values of $\mathbf{k}$.)

To compare results against a reference, we create a reference vector $T^{\text{ref}}_{\mathbf{jk}}$ using an extremely long (25 CPU-minutes) integration time and then adopt two metrics. The first, which we call the *relative Euclidean distance* (hereafter, RED) is the

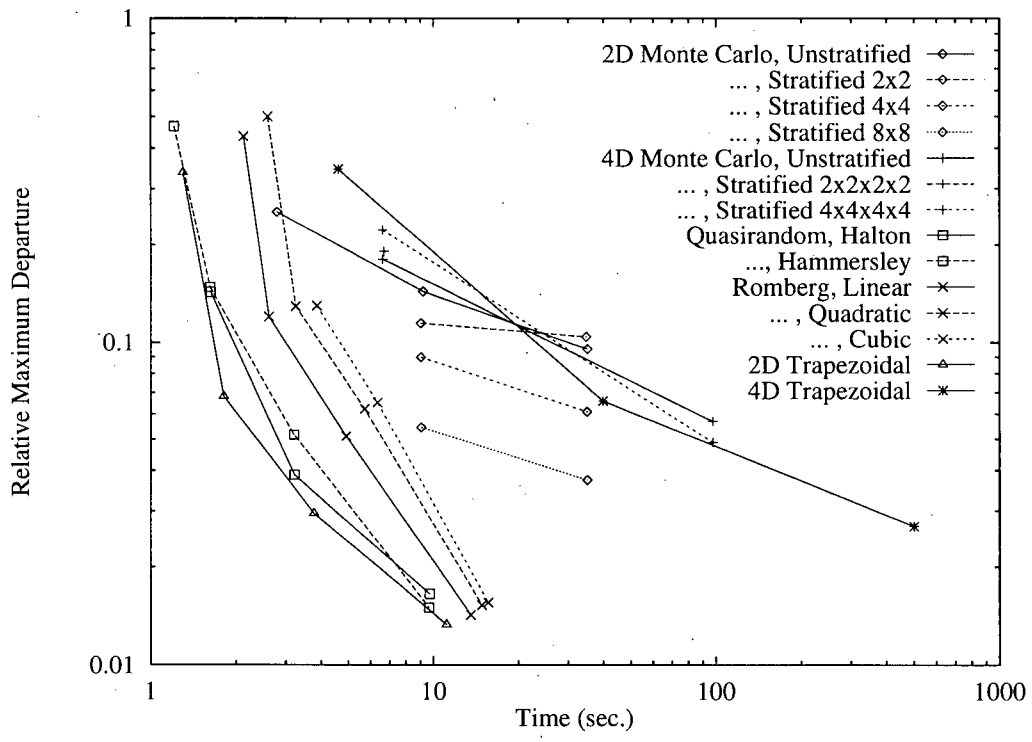

Figure 6.6: Relative Maximum Departure vs. Time for Different Integration Techniques and Numbers of Samples

Euclidean distance (L^2 norm) of $T_{\mathbf{jk}}$ from $T_{\mathbf{jk}}^{\mathrm{ref}}$ divided by the magnitude of $T_{\mathbf{jk}}^{\mathrm{ref}}$:

$$\frac{\sqrt{\sum_{\mathbf{k}} \left(T_{\mathbf{jk}} - T_{\mathbf{jk}}^{\mathrm{ref}}\right)^2}}{\sqrt{\sum_{\mathbf{k}} \left(T_{\mathbf{jk}}^{\mathrm{ref}}\right)^2}}. \tag{6.9}$$

Figure 6.5 shows how the RED varies with integration time[8] and N_{samp} for the various techniques. Note that we allow the degree of extrapolation for Romberg integration to vary from linear to quadratic to cubic. The times shown are for an IBM Model RS/6000 POWERserver 560 workstation.

[8]Rather than N_{samp}, time is the appropriate abscissa here. Since the time required to evaluate the integrand varies with dimensionality, fewer samples do not necessarily imply faster computation.

The second metric, which we call *relative maximum departure* (hereafter, RMD), is more conservative. It is the maximum absolute difference between $T_{\mathbf{jk}}$ and $T^{\text{ref}}_{\mathbf{jk}}$ (L_∞ norm) divided by the maximum absolute value of $T^{\text{ref}}_{\mathbf{jk}}$:

$$\frac{\max_{\mathbf{k}} \left| T_{\mathbf{jk}} - T^{\text{ref}}_{\mathbf{jk}} \right|}{\max_{\mathbf{k}} \left| T^{\text{ref}}_{\mathbf{jk}} \right|}. \tag{6.10}$$

Figure 6.6 shows how the RMD varies for the same parameters as Figure 6.5.

From these plots, we can draw several conclusions:

- The two figures are qualitatively similar: a method that does well by one metric generally does well by the other. This gives us some confidencé that these metrics are valid.
- Stratification improves Monte Carlo results. (This comes as no surprise.)
- The time required for Monte Carlo methods being linear in N_{samp}, they all approximate the expected $O(N_{\text{samp}}^{-1/2})$ behaviour.
- For a given integration time, the 4D trapezoidal method does generally worse than the other methods.
- Increasing the degree of extrapolation for Romberg (2D) integration generally makes matters worse. This is presumably a result of the discontinuities in the integrand we discussed above.
- Quasi-random methods give similar and comparatively good results. This reinforces the findings of Keller [39].

The most surprising observation, however, is how well the straightforward 2D trapezoidal rule does. For the RED metric, it is comparable with the best of the other

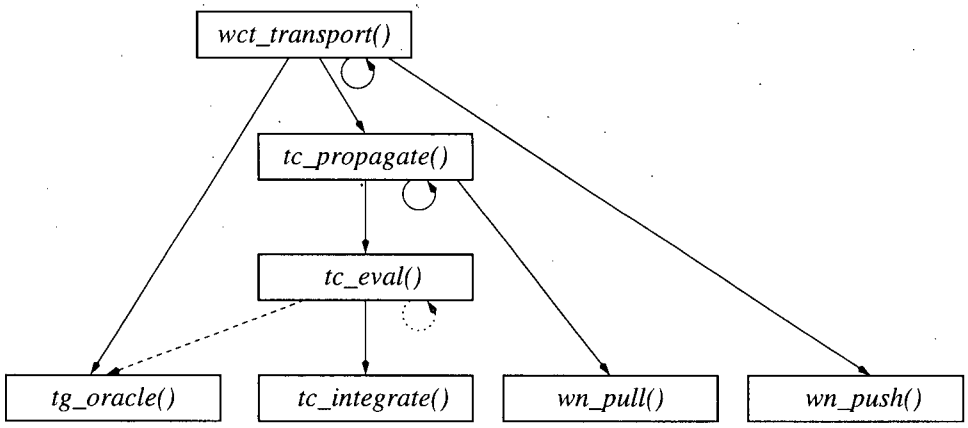

Figure 6.7: Simplified Functional Decomposition for Wavelet Radiative Transfer

methods, quasi-random, and for the RMD metric it does noticeably better for short integration times.

6.11 Functional Decomposition of *WRT*

Figure 6.7 shows a simplified functional decomposition for *WRT*. In this section, we will discuss each block individually and describe how it implements the ideas presented in Chapter 5. We will proceed in a bottom-up sequence.

6.11.1 *tg_oracle()*

This function is a geometric query function. Given a *TransportGeometry* and source and destination *WaveletIndices*, it performs several fast query functions.

- If either source or destination wavelet directional supports lie entirely outside the UDC, they cannot interact.
- Projecting the 16 vertices of the destination support hypercube project onto

the plane that contains the source, the vertices can be classified by the "fast reject" part of a four-dimensional version of the Cohen-Sutherland line clipping algorithm (see Foley, et al. [18], for example).

- If all 16 destination vertices map into points contained within the source support, we assume that the entirely of the destination support does, so an exact computation of the inner product (which is then proportional to the volume of the destination support hypercube) is possible.

The latter two items require an assumption that the projected destination support vertices define a convex hull for the entire projected support. As we noted in Section 6.9.2, this is not in general the case. Nevertheless, by disabling this oracle and comparing results, we have found empirically that it is a reasonable approximation, at least for oracular purposes.

6.11.2 *tc_integrate()*

This function is responsible for performing the actual integration. It is capable of using any of the schemes described in Section 6.10.

6.11.3 *wn_pull()* and *wn_push()*

These functions act on individual nodes of a WCT to perform, respectively, a single-level, four-dimensional (i.e. 16 value), Haar analysis (i.e., forward transform or "pull"[9]) or synthesis (i.e., inverse transform or "push").

[9]The useful "pull" and "push" terminology is from Hanrahan, et al. [32].

```
tc_propagate(WaveletIndex wiS, double bS[], WaveletIndex wiD,
             TransportGeometry tgSD, WaveletCoefficientTree wctD)
    if wiD.l == maximum wavelet level + 1
        return tc_eval(wiS, wiD, tgSD)
    else if tg_oracle(wiS, wiD, tgSD) gives exact result
        return it
    else
        for each child wiDChild of wiD
            tc[wiDChild] = tc_propagate(wiS, bS, wiDChild, tgSD, wctD)
        for each channel chan                                          10
            for each child wiDChild of wiD
                wn[chan][wiDChild] = bS[chan] * tc[wiDChild]
            wn_pull(wn)
            if wiD.l is not the top destination level
                wn[chan][wiD.nu = 0] = 0
        wctD[wiD] = wn
```

Figure 6.8: *tc_propagate()* Pseudocode

6.11.4 *tc_eval()*

Given source and destination wavelet indices, this function computes a single transport coefficient. It will call itself recursively if either source or destination basis selectors are nonzero, although as it is currently used in *WRT*, *tc_eval()* is only called to evaluate pure smoothing coefficients[10]. At the user's request, it will call *tg_oracle()* to attempt to find an alternative to numerical quadrature. Otherwise, it will call *tg_integrate()* to perform that quadrature.

6.11.5 *tc_propagate()*

This function propagates a single source wavelet coefficient to a destination wavelet coefficient tree *wctD*. Figure 6.8 shows its pseudocode. Like *tc_eval()*, it will work with a nonzero source basis selector, but in the context of *WRT* is only called upon with pure source smoothing coefficients. It is important to note that *wctD* will

[10]The ability to work with non-zero basis selectors is used in self-test mode.

contain no pure smoothing coefficients except at a specified top level. The source pure smoothing coefficients corresponding to *wiS* are passed to the function as *b0[]*.

There are three alternatives:

- If the destination wavelet is one more than the maximum wavelet level, it returns the smoothing coefficient it gets from *tc_eval()*.
- If *tg_oracle()* gives an exact result (often zero), it returns that result. Note that *wctD* does not need to be updated in this case, since the exact result is either zero or a pure smoothing result below the top level, neither of which requires storage.
- Otherwise (and most importantly) the function applies itself recursively to the 16 child indices of *wiD*, collecting the 16 returned smoothing coefficients. For each channel, it then multiplies each coefficient by the source coefficient for that channel, and uses *wn_pull()* to apply a "pull" to convert the result to wavelet coefficients. If *wiD.l* is not the top destination level, the pure smoothing coefficient is redundant and is set to zero. The result is then ready to be stored in *wctD*.

6.11.6 *wct_transport()*

Figure 6.9 shows the pseudocode for this function, which transports an entire source wavelet coefficient tree *wctS* to a destination wavelet coefficient tree *wctD*.

The function first uses *tg_oracle()* to reject impossible interactions. If there is a node *wn*, indexed by *wiS* in *wctS*, it will call itself recursively to descend the tree. First, though, to the pure smoothing coefficients of *wn*, it adds *bS0[]* (which is zero when this function is initially called before recursion). It then invokes *wn_push()* to

```
wct_transport( WaveletIndex wiS, double bS0[], WaveletCoefficientTree wctS,
               TransportGeometry tgSD, WaveletIndex wiDTop,
               WaveletCoefficientTree wctD):
    if tg_oracle(wiS, tgSD, wiDTop) says an interaction is not possible
        return
    wn = wctS[wiS]
    if wn exists,
        for each channel chan,
            wn[chan][0] = wn[chan][0] + bS0[chan]
            wn_push(wn)                                              10
        for each child wiSChild of wiS
            wct_transport(wiSChild, wn[*][wiSChild], wctS, tgSD, wiDTop, wctD)
    else
        tc_propagate(wiS, bS0, wiDTop, tgSD, wctD)
```

Figure 6.9: *wct_transport()* Pseudocode

convert the wavelet coefficients to finer pure smoothing coefficients at the level of the children of *wiS*. *wct_transport()* then applies itself recursively to each of these children, passing the elements of *wn* as new values of *bS0[]*.

If *wn* does not exist in *wctS*, there are no nodes in *wctS* at or below *wiS*, so the only thing that needs to be propagated is the pure smoothing value *bS0[]*. *wct_transport()* calls *tc_propagate()* to do this. The change of prefix is an indication that below this call level, we are no longer concerned with a source WCT, but with individual pure smoothing (multichannel) coefficient vectors.

6.12 Example of Transport

For a test configuration, we imagine light shining through the square stained glass window shown in Figure 6.11.

As illustrated in Figure 6.10, the incident light shines down with a distribution peaking at an angle of 45° from the horizontal, diffused by the glass according to a distribution proportional to the 4th power of the cosine of the angle between

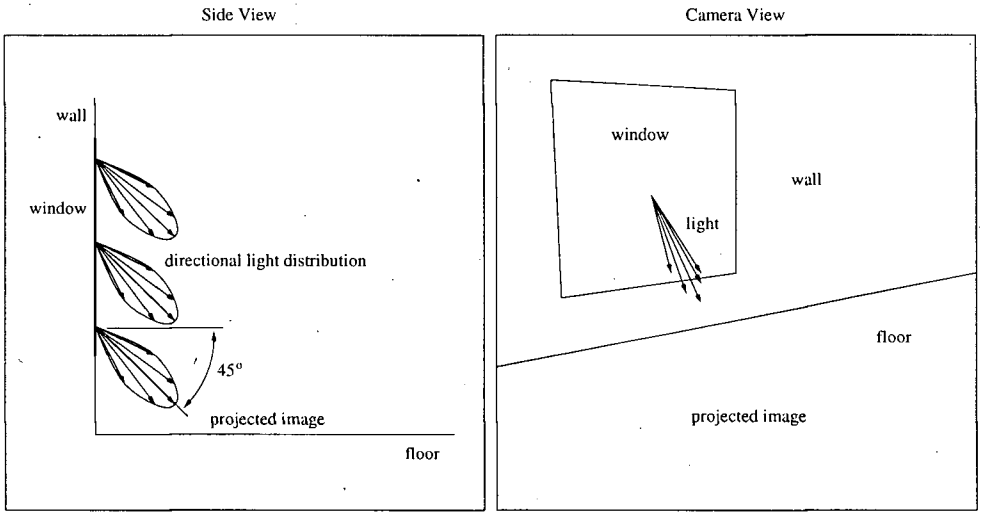

Figure 6.10: Geometry Used for Example of Transport

the propagation direction and the peak direction. The light is transported to the floor and is collected there.

Figure 6.12 shows the complete, inversely-transformed 4D results. Each of the small images represents the spatial variation of radiance in the fixed direction given by the image's position in the matrix.

Figure 6.13 is a detailed view of the brightest part of Figure 6.12. This computation of $32 \times 32 \times 32 \times 32$ coefficients (in red, green, and blue channels) compressed by 95% required 12.4 hours of CPU time on an IBM RS/6000 POWERserver 560 workstation.

Needless to say, this is impractical for a single frame, but the result is reusable.

Figure 6.14 shows several frames generated with an otherwise conventional raytracer modified to treat a wavelet radiance distribution as a "4-D texture". Each frame is generated from a different camera position. Given the diffuse nature of

Figure 6.11: A Stained Glass Window. This is the spatial component of radiance for the test configuration.

the problem, the resulting images look adequate and contain none of the "noise" common to the usual Monte Carlo approach to this sort of problem.

6.13 Example of Radiance Compression

We can use the test scene to illustrate the effects of compression discussed in Section 6.4. Figure 6.15 shows the effect of compressing nodes of the wavelet representation of the radiance by differing ratios prior to transport in the above example. It is evident that considerable compression is possible. Although beyond the scope of this thesis, more sophisticated reconstruction techniques could be applied to remove the artifacts visible for high compression.

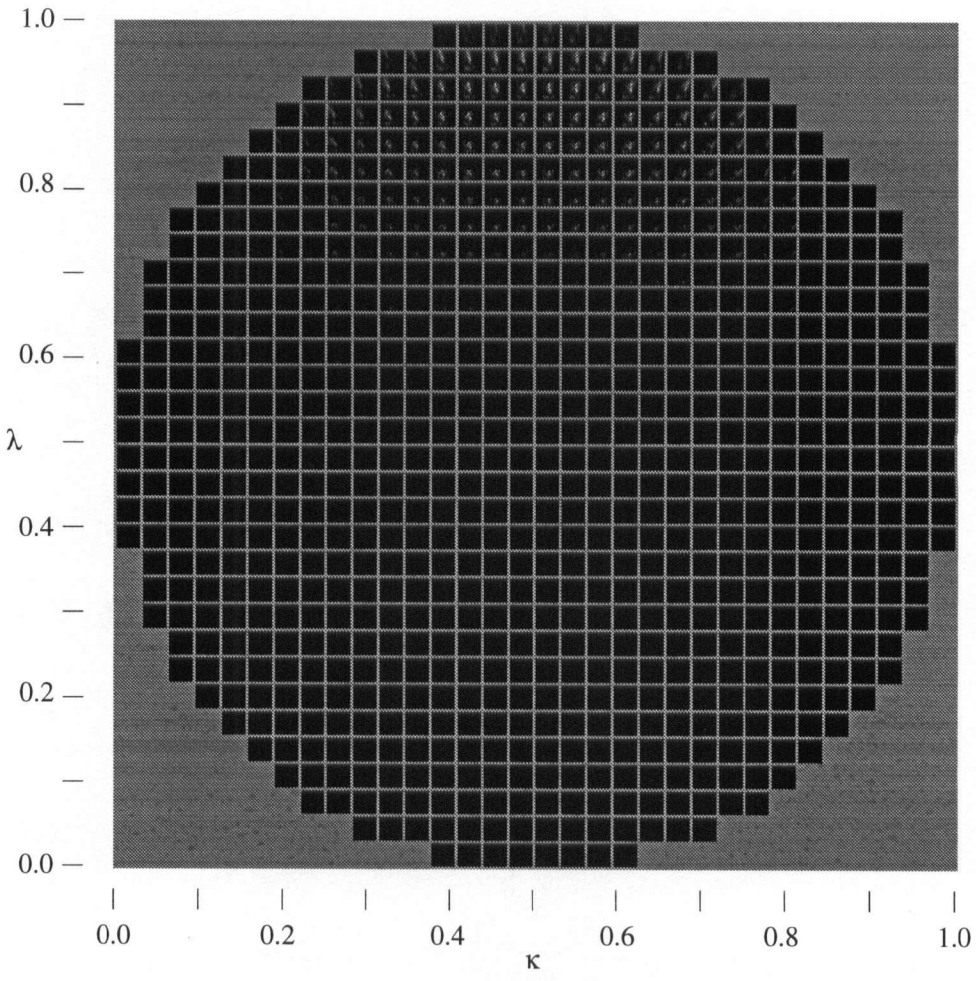

Figure 6.12: Four-Dimensional Results of Wavelet Radiative Transport

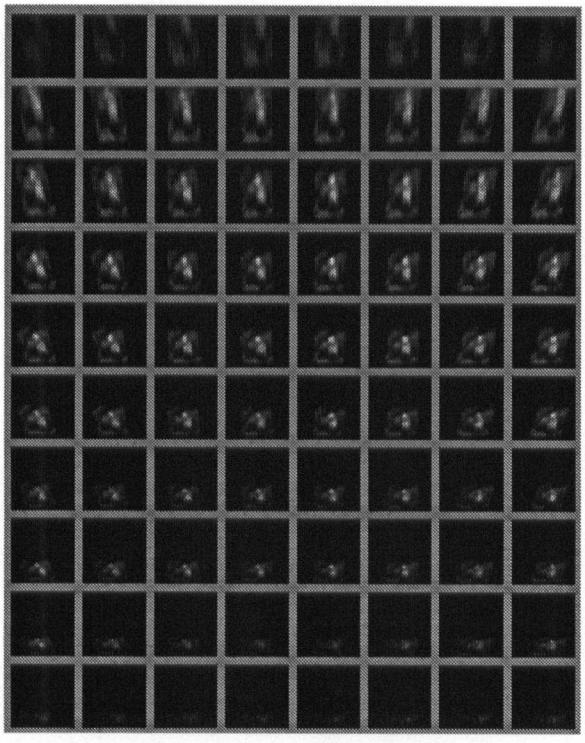

Figure 6.13: Detailed View of the Brightest Part of Figure 6.12

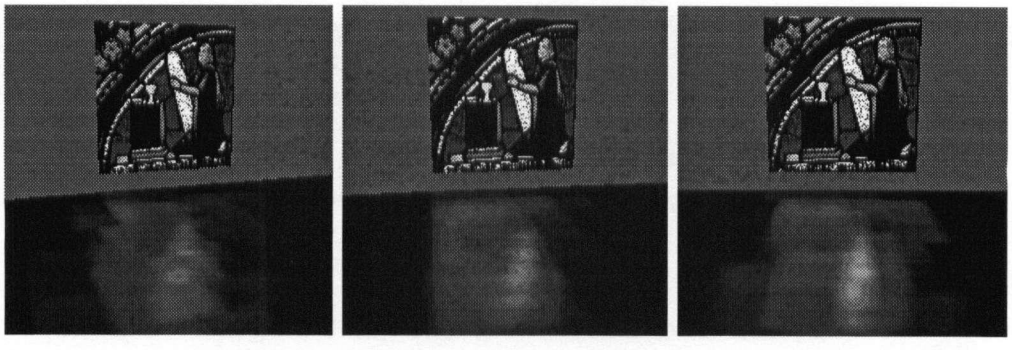

Figure 6.14: Example of Wavelet Transport. The resulting wavelet representation of radiance shown in Figure 6.12 can be sampled from different directions.

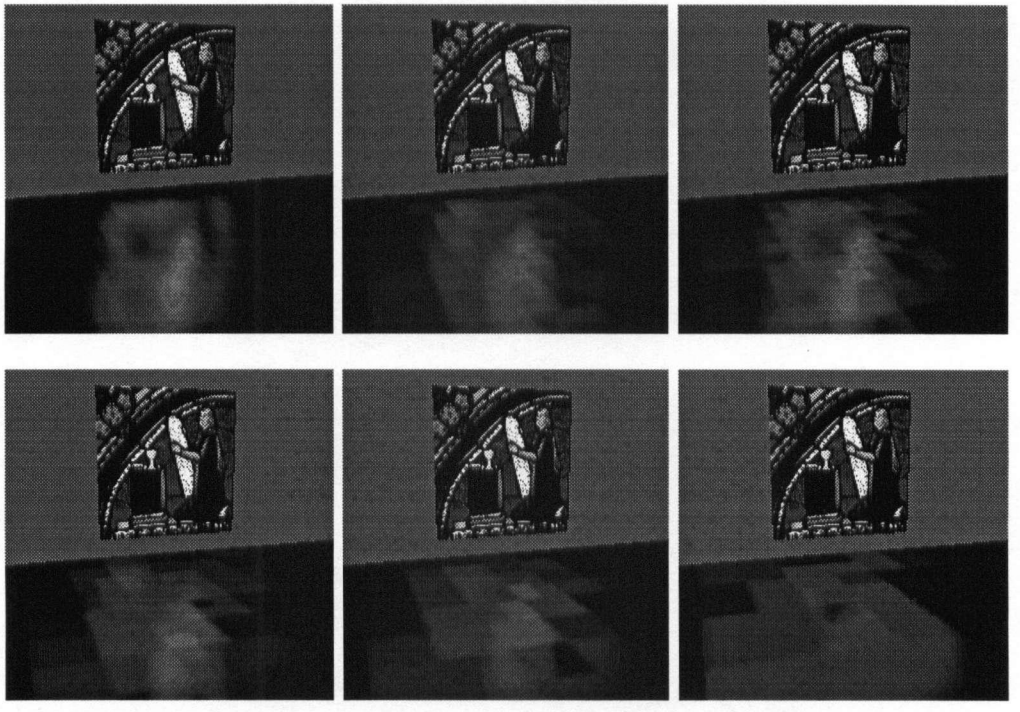

Figure 6.15: Effect of Compressing the Wavelet Radiance Distribution by (left-to-right and top-to-bottom) 2:1, 20:1, 200:1, 1000:1, 2000:1, and 10000:1

Chapter 7

Lucifer Implementation and Practicum

This chapter describes the implementation of the *Lucifer* algorithm, which we described in Chapter 3. This brings together all previous threads of discussion. It uses both the theory of WRT described in Chapter 5 and its implementation, described in Chapter 6. In addition, as we discussed in Chapter 3, in order for *Lucifer* to converge (theoretically), it is necessary to make use of the energy-conserving shaders described in Chapter 4.

As mentioned in Chapter 6, *Lucifer* makes use of all the classes that WRT uses. In this section, we will describe those classes that are unique to *Lucifer*. We will then describe the implementation choices we've made, cover a functional decomposition of it, and conclude with an example.

7.1 The *Cell* Class

Figure 7.1 shows the `typedef` we use to represent cells.

```
typedef struct _Cell {
    BoundingBox bx;
    struct _Cell *parent;
    int lvl;
    int priority;
    WaveletCoefficientTree *wct[6];
    bool isLeaf;
    union {
        struct {
            int nObj;
            Object **obj;
            bool isEmShot;
        } leaf;
        struct {
            struct _Cell *child[8];
        } nonleaf;
    } u_is;
} Cell;
```

Figure 7.1: The *Cell* typedef

If *cl* is an instance of *Cell*, *cl.bx* is its bounding box, *cl.parent* is its parent cell (or NULL if it is the root cell), *cl.lvl* is the cell level (0 for the root cell), *cl.priority* is the cell's priority used for the priority queue described in Section 3.5, *cl.wct[]* is an array of six WCTs, one for each wall of the cell, *cl.isLeaf* is TRUE if the cell is a leaf cell and FALSE if it is not.

If *cl* is a leaf cell, *cl.u_is.leaf.nObj* is the number of objects it contains and *cl.u_is.leaf.obj* is an array of (pointers to) those objects. A flag *cl.u_is.leaf.isEmShot* is set depending on whether or not emissive objects within the leaf cell have had their emissivities transferred to the cell walls (i.e. "shot"). Notice that this flag is associated with an cell rather than an object. This allows emissive objects to span cells. As we shall see below, it is the responsibility of *wct_transport()* to assure that only that part of an object's emissivity contained within the cell being balanced gets transferred to the walls.

If *cl* is not a leaf cell, *cl.u_is.nonleaf.child* is an array of (pointers to) its children.

7.2 Bidirectional Reflectance and Transmittance Distribution Functions

Like a WCT, the wavelet representation of a BRDF or BTDF is a nonstandard multiresolution 4-tensor. After the work of Lalonde and Fournier [42], we can use the same object as described in Section 6.4 with a slight reinterpretation of the meaning of the components of the wavelet indices (see Section 6.2).

To represent a BRDF or BTDF, we take the first two dimensions of the wavelet index (i.e., the ones referred to as *wi.m[0]* and *wi.m[1]*) to refer to the incident directional components of the distribution and the last two dimensions (*wi.m[2]* and *wi.m[3]*) to refer to the reflected or transmitted directional components.

As some time in the future, we may wish to allow for positional variation of BRDFs and BTDFs by using a multiresolution 6-tensor that adds the two additional surface positional components. It is even possible to allow for such phenomena as subsurface scattering (cf. Hanrahan and Krueger [31]) by using (in its most general form) a multiresolution 8-tensor.

7.3 Implementation Choices

For reasons given in Section 6.8, we have chosen to use Haar wavelets both for *WRT* and for *Lucifer*. In this section, we will discuss the other choices we've made for the *Lucifer* renderer in the context of Section 2.5.

7.3.1 Integration Scheme

Considering the results for *WRT* found in Section 6.10, it is tempting to favour use of the 2D trapezoidal method in transport coefficient computation.

However, doing so for *Lucifer* revealed a lack of robustness. Accuracy dropped considerably when the geometry departed from the simple parallel unit square geometry used for the above results. Monte Carlo and quasirandom methods did not exhibit this problem.

For this reason, we chose a quasirandom integration method in *Lucifer*. It remains to be seen, however, if other methods are possible which might be more suitable for an integrand with such discontinuities as we have.

7.3.2 Storing Transport Coefficients

Chapter 6, which considered radiative transport between two polygons, did not require the storage of transport coefficients. In other situations, however, storing them may be desirable. In this section we will outline the considerations to be made in deciding whether or not they should be stored, and if so, how?

Like form factors in radiosity, the transport coefficients are dependent only upon geometry. If the same geometry occurs in several different instances, precomputing the transport coefficients means they only have to be computed once.

Counterbalancing the argument for storing coefficients is the observation that, assuming wavelet compressibility, $b^s_{\mathbf{j}}$ is sparse. This means that only those elements of $T_{\mathbf{jk}}$ for which we have nonzero $b^s_{\mathbf{j}}$ values need to be computed.

If we decide to store transport coefficients, we face another problem: $T_{\mathbf{jk}}$ is large. If $l_{\max}$ is the maximum level of resolution on the source surface, there are $16^{l_{\max}+1}$ possible values of $\mathbf{j}$ (for Haar). Assuming the same resolution is required

for the destination surface, $T_{\mathbf{jk}}$ would require $256^{l_{\max}+1}$ elements. If represented as a conventional array, $l_{\max} = 3$ would exhaust a 32-bit address space!

Fortunately, $T_{\mathbf{jk}}$ is also sparse in this sense: for a given $\mathbf{j}$, the number of destination basis functions $B_{\mathbf{k}}$ that it affects is much smaller than the number of possible values of $\mathbf{k}$. If we do decide to put $T_{\mathbf{jk}}$ in a lookup table, that table will have to be sparse, in which case the nodal compression strategies we discussed in Section 6.4.3 would also apply to stored transport coefficients.

7.3.3 Luminaire Model

The most natural luminaires to implement in *Lucifer* are area light sources, the reason being that since we are collecting, interacting, and re-radiating WCTs on surfaces, it is simple to add an emissive WCT prior to re-radiation. Because they are represented by WCTs, these area light sources are much more flexible than the usual area light sources. In fact, they are equivalent in functionality to Levoy and Hanrahan's [44] "light fields": Any light distribution within the limit of resolution can be represented. This is truly "light though a window".

It would not be difficult to add point and directional light sources to *Lucifer*. A point light source would be treated like an object. When a cell contained a single point light source, its light could be discretized and wavelet compressed on the cell walls. Thereafter, it would be indistinguishable from any other light source. Linear luminaires could be treated similarly, with segmentation done when they spanned multiple cells, as could directional luminaires coming from within the root cell (as from a collimated source).

Directional light sources coming from infinity (approximating sunlight, for instance) could be treated as light incident on the root cell, with the light discretized

and wavelet compressed on the root cell's walls. The fact that a directional light source is of uniform radiance in a single direction would lead to a very compact representation.

7.3.4 Object Model

Our initial object models in *Lucifer* are polygonal. Other parametric forms are possible, but the difficulty of transport coefficient computation (see Equation 5.27) dictates that the geometry for this initial effort be as simple as possible.

7.3.5 Illumination Model

Lucifer's illumination model is very flexible: any BRDF or BTDF that can be represented as in Equation 5.33. The easy way to do this is to sample any BRDF or BTDF in four-dimensional Nusselt coordinates at any desired resolution, perform a (four-dimensional) Haar transform on the samples, and (for efficiency) threshold the result before storing. This only has to be done once for each distinct material and may be done prior to rendering.

7.3.6 Shading Model

It is possible to create an image from *Lucifer* using WRT directly. We can collect light on a surface WCT placed anywhere in the scene. This would have the advantage of being usable as a light field for additional rendering. The drawback, of course, is that it would not be a very good looking image, primarily because it would not be practical to render such an image at a high enough resolution to be "visually pleasing". Just as we noted in Section 2.5.7.4, the criteria for an acceptable image differ from those for physically-correct global illumination.

For this reason, as part of our "final gather" (see Section 2.5.7.4) we will make use of *FIAT*'s "light update buffer" . This is a conventional raytraced image buffer whose pixel values are determined by casting primary rays onto the emissive input and *Lucifer*-generated WCTs attached to objects in the scene. Whenever a cell is updated, all pixels whose primary rays intersect objects within that cell have their radiances incremented.

7.3.7 Surface Model

In this implementation of *Lucifer*, we assume that reflective properties of objects do not vary with position: i.e., that texturing is constant. This means that our BRDFs and BTDFs are functions only of incident and reflected or transmitted directions. At some point in the future, it would be a natural extension to allow two additional positional degrees of freedom, but our need to reduce storage requirements demands that we defer a full treatment of texturing.

7.4 Surface Interaction with Haar Wavelets

In Section 6.8, we described the benefits of Haar wavelets for *WRT*. In this section, we will demonstrate an additional benefit that pertains to *Lucifer*: the fast computation of surface interaction.

Recall from Equation 5.47 that we can compute surface interaction directly from wavelet representations of the incident radiance and the BRDF or BTDF if we can compute the multiresolution delta tensors $\Delta^{\sigma\tau}_{\mathbf{jk}}$. For most wavelets, these tensors need to be computed numerically, since they rely upon such things as the inner product of a smoothing function and its dual of differing level. As we will shortly see, orthogonality helps somewhat, but not completely because of the multidimensional

nature of the inner products.

$\Delta_{\mathbf{jk}}^{\sigma\tau}$ tensors for Haar wavelets, on the other hand, *do* have a closed form representation:

$$\Delta_{\mathbf{jk}}^{\sigma\tau} = Q^{-\frac{1}{2}} \begin{cases} \sum_{m''=0}^{Q-1} \delta_{(m_\sigma^j + m'')(Q m_\tau^k)} & \nu_{\langle\sigma\rangle}^j \nu_{\langle\tau\rangle}^k = 00 \wedge l^j \geq l^k \\ \sum_{m''=0}^{Q-1} \delta_{(Q m_\sigma^j)(m_\tau^k + m'')} & \nu_{\langle\sigma\rangle}^j \nu_{\langle\tau\rangle}^k = 00 \wedge l^j \leq l^k \\ \sum_{m''=0}^{\frac{Q}{2}-1} \delta_{(m_\sigma^j + m'')(Q m_\tau^k)} & \\ \quad - \sum_{m''=\frac{Q}{2}-1}^{Q-1} \delta_{(m_\sigma^j + m'')(Q m_\tau^k)} & \nu_{\langle\sigma\rangle}^j \nu_{\langle\tau\rangle}^k = 01 \wedge l^j > l^k \\ 0 & \nu_{\langle\sigma\rangle}^j \nu_{\langle\tau\rangle}^k = 01 \wedge l^j \leq l^k \\ 0 & \nu_{\langle\sigma\rangle}^j \nu_{\langle\tau\rangle}^k = 10 \wedge l^j \geq l^k \\ \sum_{m''=0}^{\frac{Q}{2}-1} \delta_{(Q m_\sigma^j)(m_\tau^k + m'')} & \\ \quad - \sum_{m''=\frac{Q}{2}-1}^{Q-1} \delta_{(Q m_\sigma^j)(m_\tau^k + m'')} & \nu_{\langle\sigma\rangle}^j \nu_{\langle\tau\rangle}^k = 10 \wedge l^j < l^k \\ \delta_{l^j l^k} \delta_{m_\sigma^j m_\tau^k} & \nu_{\langle\sigma\rangle}^j \nu_{\langle\tau\rangle}^k = 11 \end{cases} \tag{7.1}$$

where $Q \equiv 2^{|l^j - l^k|}$. Of the seven cases here, the fourth, fifth, and seventh would be true for any wavelet, not just Haar.

Equations 5.47 and 7.1 tell us how to evaluate the inner products for a given $\mathbf{j}$, $\mathbf{k}$, and $\mathbf{n}$, but that is not what is required algorithmically. We start with sparse sets of coefficients $\left\{f_{\mathbf{j}}^r\right\}$ and $\{b_{\mathbf{k}}\}$ and want to generate a new set $\{b_{\mathbf{n}}\}$. We need to organize our implementation of Equation 5.47 in particular so that it performs the mapping $(\mathbf{j}, \mathbf{k}) \rightarrow \{\mathbf{n}, b_{\mathbf{n}}\}$ using 7.1 to eliminate as much as possible the generation of $\mathbf{n}$ values which do not contribute. The pseudocode for this is lengthy and therefore included as Appendix B.

Future work should consider the application of these techniques to the general problem of multidimensional inner products:

$$f(\mathbf{x}, \mathbf{y}) = \langle g(\mathbf{x}, \cdot) \mid h(\mathbf{y}, \cdot) \rangle_{\mathbf{z}} . \tag{7.2}$$

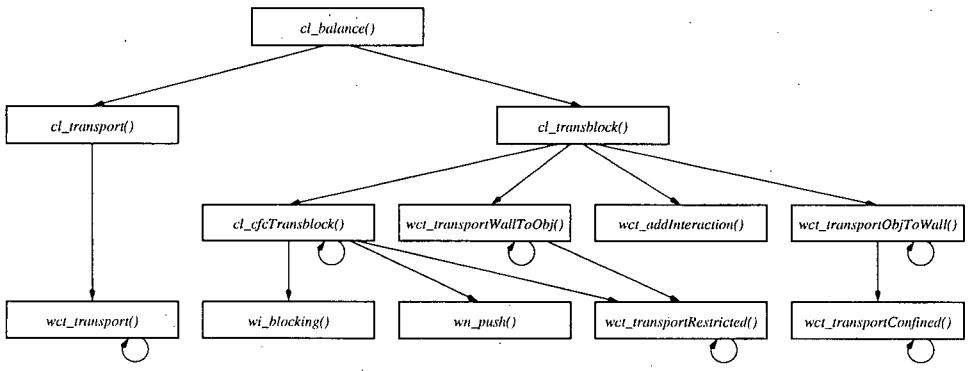

Figure 7.2: Simplified Functional Decomposition for *Lucifer*

Convolution is a special case of this, although we expect it to be more useful in other cases such as spatially-variant filtering.

7.5 Functional Decomposition of *Lucifer*

Figure 7.2 shows a simplified functional decomposition for Lucifer.. In this section, we will describe these building blocks individually. As in Section 6.11, we proceed in a bottom-up order.

7.5.1 *wct_transportConfined()*

This is the same as *wct_transport()*, with one enhancement: the ability for the user to confine all transport computations to the bounding box of a given cell. This allows *Lucifer* to deal with objects that span cell boundaries, the need for which was described in Section 7.1.

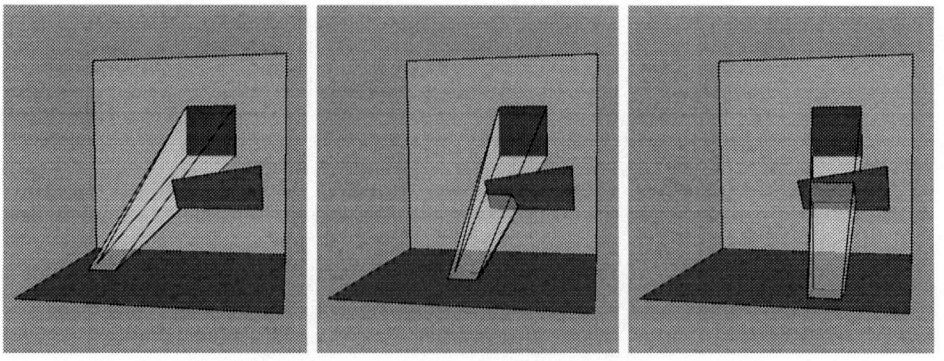

Figure 7.3: Unblocked, Partially Blocked, and Completely Blocked Propagation. The source wavelet support is red, the destination wavelet support is green, and the blocking rectangle is blue.

7.5.2 *wct_transportRestricted()*

This is another modification of *wct_transport()* which allows the caller to restrict both source and destination WCTs to subtrees of specified wavelet indices. Since it is always used for wall-to-wall transport, the confinement to a cell's bounding box is implicit.

7.5.3 *wi_blocking()*

This function determines the degree to which an object is capable of blocking the transfer of radiance from the support of one Haar wavelet coefficient (and its children) to another (and its children). Figure 7.3 illustrates how blocking is determined. Given a source coefficient (on the vertical wall), and a destination coefficient (on the horizontal wall), the function constructs the "blocking hull": the polyhedron which is the convex hull bounded by their (square) spatial supports, which we may refer to as the "end caps".

Once the hull is constructed, *wi_blocking()* determines the nature of the in-

tersection of the hull and the object. If the object lies entirely outside the blocking hull, every point in the source coefficient support can "see" every point in the destination support, so the function returns *NONE*. If the object "slices" the blocking hull into two regions, each of which wholly contains a single end cap, no point in the source support can "see" a point in the destination support, so *wi_blocking()* returns *FULL*. Otherwise, at least one source point can see at least one destination point, so *blockingType()* returns *PARTIAL*.

This query does not take into account the *directional* components of the supports: that is left for the actual transport computation done by *wct_transport()* and its variants.

7.5.4 *cl_cfcTransblock()*

Given source and destination cell walls, this function, the pseudocode for which is in Figure 7.5, is responsible for transporting that part of the source wall's WCT that is not obstructed by the objects contained within the cell to the destination wall.

For each object in the cell, it calls *wi_blocking()*. If any single object totally blocks transport, no transport takes place, so it returns. If no object blocks transport at all, it calls *wct_transportRestricted()* to perform the actual transport and returns.

Otherwise, there is partial blocking, so *cl_cfcTransblock()* acts recursively. If either the source or destination wavelet indices are above the maximum wavelet level, it refines the coarser of them and invokes itself on the children. If the source index is the coarser one, it also needs to push the source WCT before recurring.

If the source and destination wavelet coefficients are at the maximum wavelet level, it assumes total blocking and does nothing. This rationale deserves justification. If there is a single object inside the cell, the choice is arbitrary as we are at the

```
cl_cfcTransblock(Cell cl, WaveletCoeffcientTree wctSWall,
        TransportGeometry tgSD, WaveletCoefficientTree wctDWall,
        WaveletIndex wiS, WaveletIndex wiD):
    blocking = NONE
    for each object obj in cl,
        switch (wi_blocking(wiS, obj, wiD))

        case NONE

        case PARTIAL                                                    10
            if blocking = NONE
                blocking = PARTIAL

        case TOTAL
            return  // no transport

    switch (blocking)

    case NONE
        wct_transportRestricted(wiS, wctSWall, tgSD, wiD, wctDWall, cl->bx)   20

    case PARTIAL
        if wiS.l and wiD.l are the maximum wavelet level
            // assume total blocking
        else if wiS.l < wiD.l
            wn = wctS[wiS]
            delete wctS[wiS]
            wn_push(wn)
            for each child wiSChild of wiS,
                wctS[wiSChild][wiSChild.nu = 0] = wn[wiSChild]          30
                cl_cfcTransblock(cl, wctSWall, tgSD, wctDWall, wiSChild, wiD)
        else
            for each child wiDChild of wiD,
                cl_cfcTransblock(cl, wctSWall, tgSD, wctDWall, wiS, wiDChild)
```

Figure 7.4: *cl_cfcTransblock()* Pseudocode

maximum resolution and expect sampling effects to show up at this level in any case. If there is more than one object in the cell, that means that the cell itself is *also* at maximum resolution. The most likely circumstance in this case is that the objects are joined as, for example, polyhedral facets. In this case, allowing transport to take place would permit "holes" in the polyhedron, which would be an undesirable

```
wct_transportWallToObj(WaveletCoefficientTable wctSWall, Cell cl)
    for each object obj contained within cl,
        if obj has a BRDF or BTDF on its upper side,
            wct_transportConfined(cl.wiTop, wctSWall, tgSD, obj.wiTop,
                    obj.wctIncomingAbove, cl.bx)
        if obj has a BRDF or BTDF on its lower side,
            wct_transportConfined(cl.wiTop, wctSWall, tgSD, obj.wiTop,
                    obj.wctIncomineBelow, cl.bx)
```

Figure 7.5: *wct_transportWallToObj()* Pseudocode

artifact. We therefore assume total blocking in this case.

7.5.5 *wct_transportWallToObj()*

Given a cell wall, this function, the pseudocode for which is in Figure 7.5, calls *wct_transportConfined()* for each object contained within the cell, restricting transport to the bounding box of the cell in case an object extends outside the cell. Since *Lucifer* allows objects to have distinct BRDFs and BTDFs on both sides of their surfaces, we need to collect incoming radiance on both sides as well.

Notice that this function does not allow for blocking of one object by another. Inter-object blocking is addressed in *Lucifer* by choosing a maximum cell division level such that the maximum number of objects in a cell is approximately one.

7.5.6 *wct_addInteraction()*

This is an implementation of Haar wavelet surface interaction as discussed in Section 7.4. Pseudocode for this is in Appendix B.

```
wct_transportObjToWall(Cell cl, TransportGeometry tgSD,
        WaveletCoefficientTree wctDWall):
    for each object obj contained within cl,
        wct_transportConfined(obj.wiTop, obj.wctOutgoingAbove, tgSD, cl.wiTop,
                wctDWall, cl.bx)
        wct_transportConfined(obj.wiTop, obj.wctOutgoingBelow, tgSD, cl.wiTop,
                wctDWall, cl.bx)
```

Figure 7.6: *wct_transportObjToWall()* Pseudocode

```
cl_transport(Cell cl):
    for each channel chan,
        bS0[chan] = 0
    wiRoot = { 0, 0, 0, 0, 0, 0 }
    for each incoming wall wS of cl,
        for each outgoing wall wD of cl,
            tgSD = tg_init(cl, wS, wD)
            wct_transport(wiRoot, bS0, cl.wctWall[wS], tgSD, wiRoot,
                    cl.wctWall[wD])
```

Figure 7.7: *cl_transport()* Pseudocode

7.5.7 *wct_transportObjToWall()*

As with *wct_transportWallToObj()* (see Section 7.5.5), this function, whose pseudocode is shown in Figure 7.6, calls *wct_transportConfined()* for each object contained within the cell, restricting transport to the bounding box of the cell in case an object extends outside the cell. In the case of radiance that has been collected on the object surfaces by *wct_transportWallToObj()*, this restriction is redundant, but it is necessary in the case of emissive objects, whose emissive-WCTs may extend outside the cell walls.

Also like *wct_transportWallToObj()*, this function allows from transport from both surfaces of the object.

```
cl_transblock(Cell cl):
    for each incoming wall wS,
        for each outgoing wall wD != wS,
            tg = tg_init(cl, wS, wD)
            cl_cfcTransblock(cl, cl.wctWall[wS], tgSD, cl.wctWall[wD],
                    wiRoot, wiRoot)
    for each incoming wall wS,
        wct_transportWallToObj(cl.wctWall[wS], cl)
    for each object obj in cl,
        obj.wctOutgoingAbove = wct_interact(obj.wctIncidAbove, obj.above.BRDF)
                + wct_interact(obj.wctIncidBelow, obj.below.BTDF)
        obj.wctOutgoingBelow = wct_interact(obj.wctIncidAbove, obj.above.BTDF)
                + wct_interact(obj.wctIncidBelow, obj.below.BRDF)
    for each outgoing wall wD,
        wct_transportObjToWall(cl, cl.wctWall[wD])
```

Figure 7.8: *cl_transblock()* Pseudocode

7.5.8 *cl_transport()*

Figure 7.7 shows the pseudocode for this function. It is quite straightforward: for every distinct pair of source and destination walls, invoke *wct_transport()*. Note that since the source and destination WCTs are by definition contained within *cl*, it is not necessary to check for confinement, so *wct_transport()* can be used directly instead of *wct_transportConfined()*.

7.5.9 *cl_transblock()*

Wavelets provide us with the ability to selectively refine a light representation so that we can realize the *transblock()* function described in Chapter 3. Figure 7.8 is the pseudocode for *cl_transblock()*.

One of two things happens to light that enters a cell containing an object: either it passes through the cell unimpeded from incoming cell wall to outgoing cell wall or it impinges on the surface of an object and is (possibly) re-radiated to an

```
cl_balance(Cell cl):
    if cl contains no objects,
        cl_transport(cl)
    else
        cl_transblock(cl)
```

Figure 7.9: *cl_balance()* Pseudocode

outgoing cell wall.

To deal with the first case, this function calls *cl_cfcTransblock()* for each distinct pair of cell walls.

To deal with the second case, it first uses *wct_transportWallToObj()* to transfer the radiance of all walls to all objects in the cell. Then it invokes *wct_interact()* to compute the outgoing object surface WCTs, allowing for two-sided objects if specified. Finally, it uses *wct_transportObjToWall()* to transfer the object surface WCTs to the outgoing walls.

7.5.10 *cl_balance()*

Figure 7.9 shows the pseudocode for this function, which is quite simple: To balance a cell, call *cl_transport()* if the cell is empty and *cl_transblock()* if it is not.

7.6 Results

To demonstrate *Lucifer*, we will use a simple example: a source emissive square illuminating a destination triangle with an optional blocking triangle between them, as shown in Figure 7.10. Directionally, the emissivity of the square is a cosine lobe similar to 6.12, but positionally, it is uniform over the square, giving the light distribution the character of a spotlight.

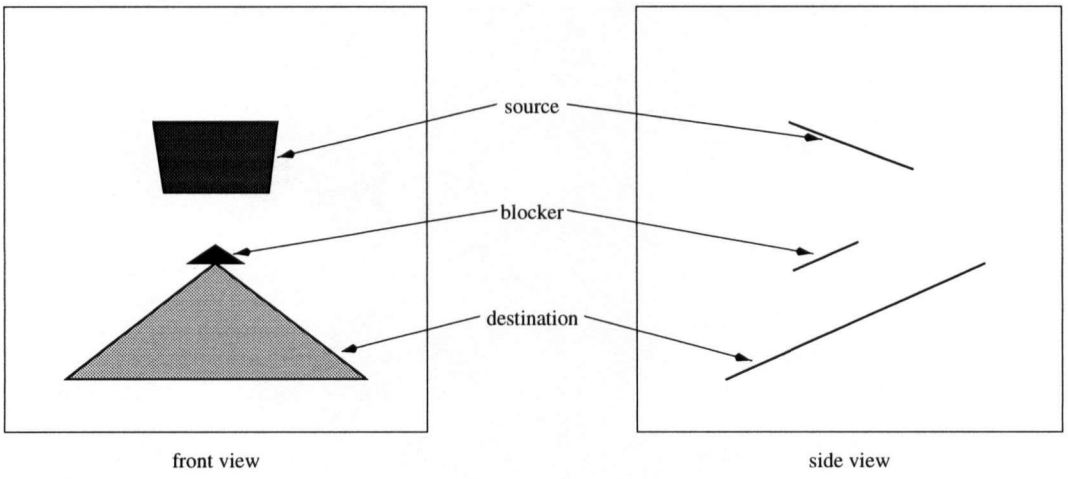

Figure 7.10: Configuration for *Lucifer* Example

Several discretionary variables determine how long *Lucifer* takes to run, how much memory it requires, and the visual quality of its output. They are:

- $l_{\max}^{\text{cell}}$, the maximum subdivision level of the cell walls.
- whether or not the blocker is in place. This allows us to test the efficiency of our blocking algorithm.
- $l_{\max}^{\text{surf}}$, the maximum subdivision level of light source and surface BRDFs. We could, in principle, select a different $l_{\max}$ for each surface and light source, but we will set them all to the same value in this example to keep the number of test cases manageable. In general, it makes sense to restrict $l_{\max}^{\text{surf}} \leq l_{\max}^{\text{cell}}$. Otherwise, the higher resolution is lost during transfer.
- whether or not the oracle is used at the lowest level as an alternative to numerical quadrature (as described in Section 6.11.4).

We have constructed several tests cases that vary these parameters, as shown

Case	Uses Blocker?	l_{max}^{surf}	Low-Level Oracle?
1	yes	2	no
2	yes	3	no
3	yes	3	yes
4	yes	4	no
5	no	2	no
6	no	3	yes
7	no	3	no

Table 7.1: Lucifer Test Cases

in Table 7.1. As it is likely to have the greatest influence on the results, we will take l_{max}^{cell} as the principal independent variable for each test case. We run each case until the unbalanced energy remaining in the scene is approximately 2% of the original luminaire energy.

Integration in each case was done using the Halton quasirandom integration method (see Section 6.10.2) with 16 samples. We used a four-dimensional scheme instead of the two-dimensional approach discussed in Section 6.10.1. Since the need to be able to restrict transport as required by *wct_transportConfined()* requires additional clipping to guarantee that the destination or projected source support rectangles lie within the cell, this increased the cost of the two-dimensional approach. In four dimensions, confining the integration requires a simple bounding box test of each point. As we will discuss below, however, the choices for integration scheme turn out not to be critical performance factors.

Lucifer allows the user to save as a sequence of frames the intermediate contents of the light update buffer while the algorithm runs. Figure 7.11 shows these for Case 4 ($l_{max}^{cell} = l_{max}^{surf} = 4$, blocker in place, no low-level oracle). In this figure, we have omitted frames that show no visible change, such as cell splits or transport through empty cells.

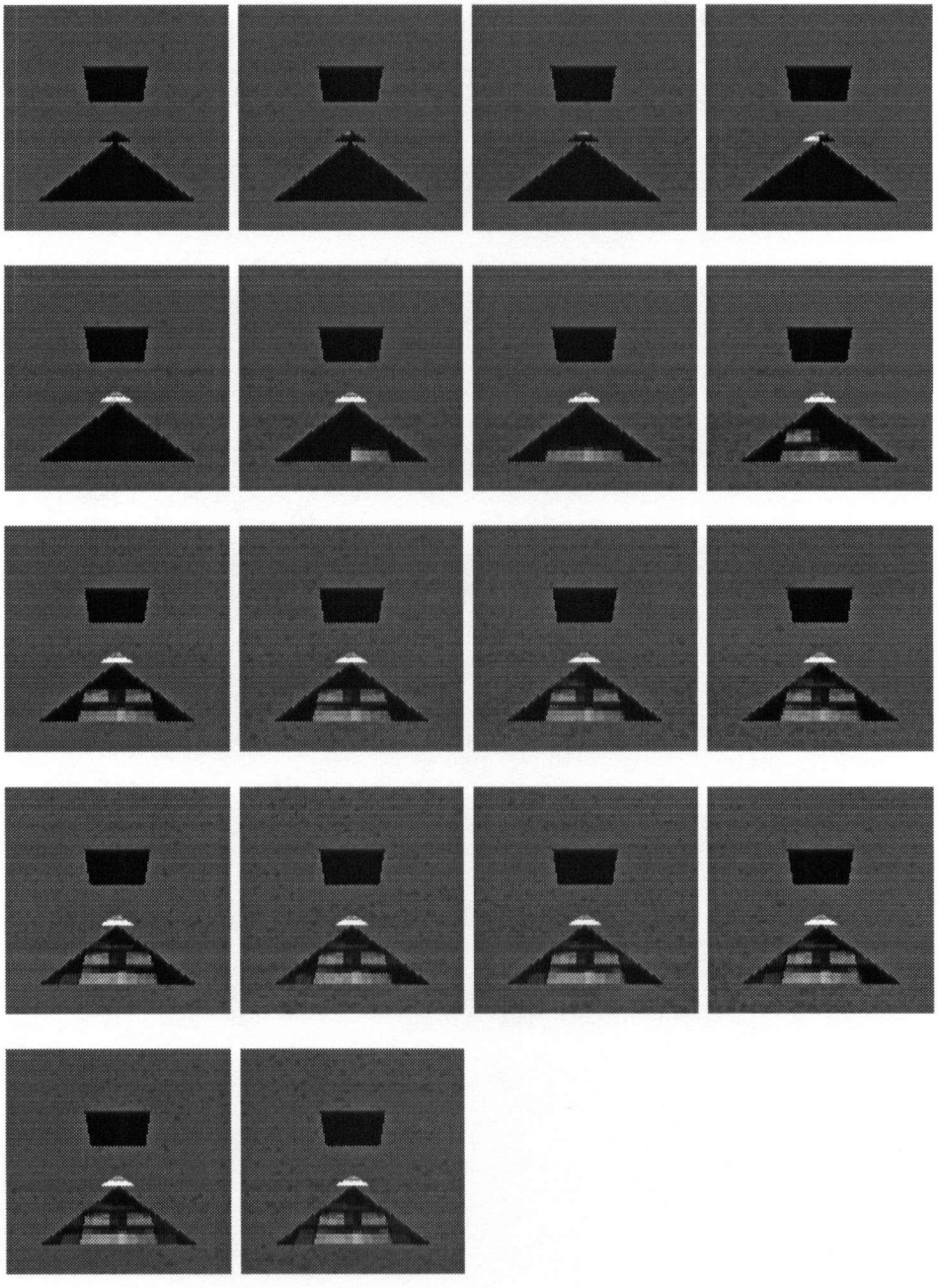

Figure 7.11: *Lucifer* Sequence. Each frame represents a non-trivial step in the progress of the *Lucifer* algorithm. This is Case 4 with $l_{\max}^{\text{cell}} = 4$.

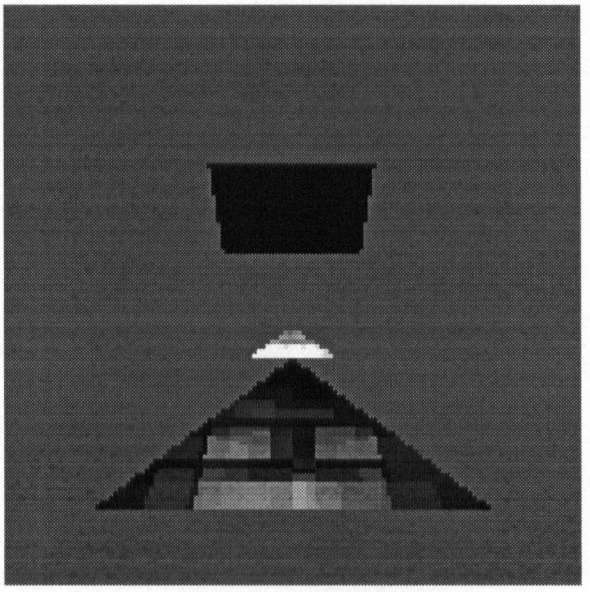

Figure 7.12: *Lucifer* Example

Figure 7.12 is an enlarged view of the final frame: the end result of *Lucifer*. There are obvious visual artifacts, but these are comparable to those produced by radiosity and other global illumination schemes in the absence of final gather (cf. Figure 4, Color Plate 7(d) in Christensen, et al. [11]).

Figures 7.13 and 7.14 show the timing and peak memory usage for all test cases examined. These were collected on a Micron Paradigm XLI workstation with a 200MHz Intel Pentium Pro CPU and 256MB of memory. Each plot shows a comparison line which is proportional to $16^{l_{\max}^{\text{cell}}}$, which would be the space complexity in the absence of wavelet compression.

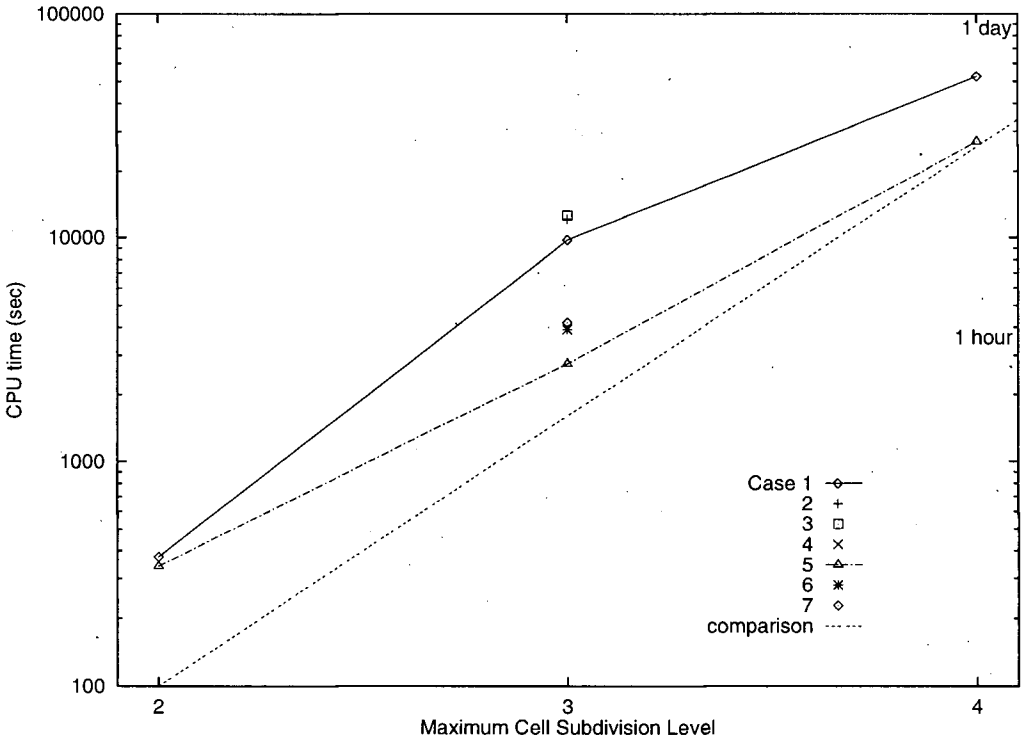

Figure 7.13: *Lucifer* CPU Time vs. $l_{\max}^{\text{cell}}$. (see Table 7.1 for test case configurations)

7.7 Analysis

In order to explain the excessive resource requirements of *Lucifer*, we instrumented the program and made several profiling runs with various sets of parameters. We made two important discoveries.

First of all, *Lucifer* spends a considerable amount of time using the oracle, even when the low-level oracle is turned off. As $l_{\max}^{\text{cell}}$ and $l_{\max}^{\text{surf}}$ increase, the mean depth of a source coefficient increases, and since the oracle must be consulted on each level, the amount of oracular work required per source coefficient goes up. Within the oracle, most of the time is spent projecting points from one surface to

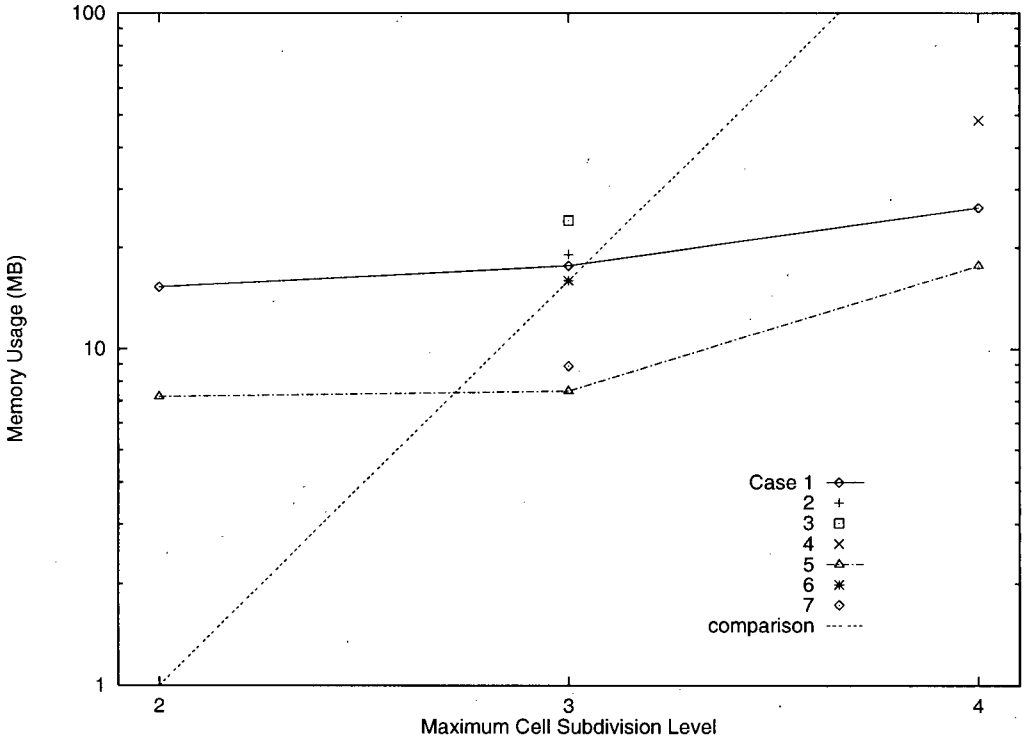

Figure 7.14: *Lucifer* Peak Memory Usage vs. $l_{\max}^{\text{cell}}$. (see Table 7.1 for test case configurations)

the other. Recall that each invocation of the oracle requires the projection of 16 points (the corners of the destination support hypercube).

After we made this observation, we tried to minimize the number of projections required by caching destination-to-source projections[1]. Unfortunately, the destination coefficient space *prior to compression* is not sparse, and the cache, even for $l_{\max}^{\text{cell}} = 4$ used an exorbitant amount of memory.

The second discovery we made is that when dealing with the largest cell, fewer than 2% of the integrations produced a non-zero result. Recall that in order

[1]Recall that integration is being done in destination space, so most projections start from there.

for an integration to be made (always at the finest pure smoothing level, $l_{\max}^{\text{cell}} + 1$ or $l_{\max}^{\text{surf}} + 1$, depending on the destination), the oracle on all coarser levels must indicate that the two coefficients can interact. This low level of accuracy could in part be due to the choice of integration scheme, but since we have observed the same phenomenon with several such schemes (particularly varying the number of samples) it is much more likely due to the nature of our oracle. Attempts to devise a more accurate one have so far been unrewarding.

As to *Lucifer*'s space requirements, as we noted above, the advantage of wavelet compression is only available after we have collected destination smoothing coefficients. We cannot compress a WCT while it is being accumulated.

There is another aspect in which compression doesn't help. Consider a wavelet source node, which can contain up to 16 wavelet coefficients (perhaps including a pure smoothing component from the node's parent). Recall from Section 6.11.6 that to transport the contents of the node, *wct_transport()* pushes the node contents and then invokes *tc_propagate()* on the resulting pure smoothing coefficients. As long as the node contains one non-pure smoothing coefficient, the amount of time required to transport the contents of the node is the same. While it reduces space requirements, compression only helps time requirements when it removes entire nodes, not individual coefficients.

Chapter 8

Conclusions and Future Work

While we have shown light-driven global illumination with wavelets to work in principle, extrapolating the *Lucifer* performance results in Section 7.6 to what would be considered a typical rendered scene ($N_{\text{obj}} \sim 1000$ and $l_{\text{max}}^{\text{cell}} \geq 6$) makes it clear that the current implementation still has a long way to go to compete with other rendering techniques such as photon maps in time efficiency, space efficiency, or visual quality (although the last might not be so bad with an appropriate final gather such as Lalonde and Fournier [42]).

Nevertheless, in implementing *Lucifer* we have learned much about the wavelet representation of radiance and have made several discoveries along the way that are worthy of note:

- We have examined the physical plausibility of several illumination models and suggested some useful new ones.
- We have shown how a wavelet representation of radiance in Nusselt coordinates leads to the simplified computation of radiative transport and other physical quantities.

- We have presented a scheme for computing transport coefficients that uses a combined geometric and numerical algorithm.
- We have investigated compression strategies for radiance and transport coefficients.
- We have developed a way to compute surface interaction that acts directly on the wavelet-transformed coefficients of an incident radiance and a BRDF or BTDF. The algorithm used for this shows promise of having applicability in more general numerical inner product computations.
- We have shown that the representation of radiance on a wall leads directly to an efficient representation of light fields.

We have spent a considerable amount of time trying to improve the speed of *Lucifer*. While we have gotten speedups of better than an order of magnitude (not including faster hardware) as a result, there is clearly still much room for improvement. Even using the oracle, the program spends far too much time generating null smoothing coefficients and (partly as a result of the oracle itself) makes too many source-to-destination (or vice-versa) projections per coefficient.

To make *Lucifer* viable, we believe the most promising direction for further study would be to move from a numerical/geometrical integration scheme to one that was entirely geometrical. The goal of this work would be to find an algorithm to determine the four-dimensional volume of the intersection of the projection of a source hypercube into destination space with a destination hypercube. Such an algorithm would speed up *Lucifer* dramatically.

The "spin-off" work mentioned above done that resulted from *Lucifer* exploration has its own set of future directions.

8.1 Fitting Plausible Shaders to Physical Data

Now that we are able to classify shaders as to their plausibility, there is a need to fit them to some real reflectance data as Larson [78] did with his own illumination model.

8.2 Fitting Plausible Shaders to Physical Data

Since *Lucifer* deals with four-dimensional radiance representations of light intrinsically, it should be possible to incorporate Levoy and Hanrahan [44] light fields or Gortler, et al. [27] lumigraphs directly into it, both as inputs and outputs.

8.3 Curved Surfaces

In this thesis, for reasons of simplicity we have considered only planar surfaces. With appropriate extensions of the parametric-to-object mappings, resulting complication of the transport computations[1], and some way to treat self-shadowing, it should be possible to use WRT to transport light from one curved parametric surface to another.

8.4 Importance

In much the same way that importance can be incorporated into other global illuminations schemes (cf. Smits, et al. [72] and Aupperle and Hanrahan [3], it would fit well into the *Lucifer* approach. We would take the observer to be an emissive

[1]To do this, we would probably have to give up on geometric integration schemes and fall back on entirely numerical ones.

source of importance (probably a point source confined to the viewing frustrum) and propagate it through cells separately from, but in the same manner as, radiance.

We would modify the algorithm of Chapter 3 so that the selection of cell to be balanced would take into account not just the cells' undistributed power, but also their (integrated) importances.

8.5 Unifying Illumination Modelling and Surface Modelling

The parameterization of BRDFs and BTDFs we have used here was four-dimensional. It is possible to incorporate the spatial variation of reflectance properties (i.e., texturing) by adding two additional dimensions to these distribution functions. This would represent the unification of illumination and surface modelling.

Adding two more positional dimensions to represent an incident-point-to-emergent-point offset could incorporate subsurface scattering. Extending our fast inner product algorithm to work in this case would be trivial.

We could even use *Lucifer* itself to generate such positional BRDFs and BTDFs on detailed microstructural models for later incorporation in larger-scale image synthesis.

Appendix A

One-Dimensional Wavelet Properties

In this appendix, we will review some of the properties of wavelets that make them particularly suitable for the representation of radiance. The standard reference on wavelets is Daubechies [16], from which much of this appendix is derived. We will confine ourselves here to one-dimensional wavelets. Section 5.3 will extend these properties to multiple dimensions.

A.1 Scaling Functions and Wavelets

Wavelets are built from *scaling functions*, which we define by enumerated dilations and translations of a base scaling function $\phi(x)$ of the form:

$$\phi_{lm}(x) = 2^{l/2}\phi(2^l x - m) \tag{A.1}$$

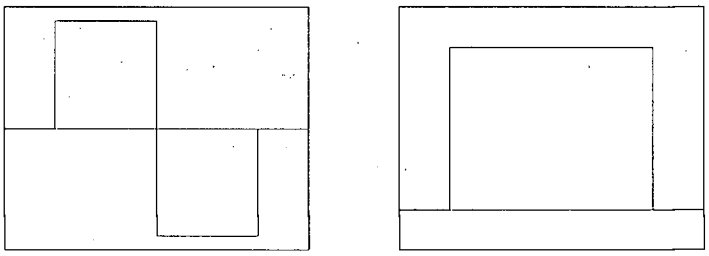

Figure A.1: Haar Wavelet and Smoothing Functions. Haar wavelets are the simplest possible wavelet and are the most commonly used wavelets in graphics. They have only one vanishing moment.

each level l corresponds to a function space V_l, which is part of a nested sequence of subspaces $\ldots \subset V_{-1} \subset V_0 \subset V_1 \subset V_2 \ldots$[1].

Scaling functions have the property that

$$\int_{-\infty}^{+\infty} \phi(x)dx \neq 0 \tag{A.2}$$

We define a wavelet function space W_l as composed of those functions that need to be added to a given space V_l to span the next finer space V_{l+1}: $V_{l+1} = V_l \oplus W_l$. The basis functions for W_l are then dilations and translations of a *mother wavelet* $\psi(x)$:

$$\psi_{lm}(x) = 2^{l/2}\psi(2^l x - m) \tag{A.3}$$

Wavelets have the property

$$\int_{-\infty}^{+\infty} \psi(x)dx = 0 \tag{A.4}$$

Figures A.1-A.3 show some commonly-used wavelets and their corresponding smoothing functions.

[1]This notation is a departure from that of Daubechies [16] that is more suitable for discrete data: an increasing subscript of V corresponds to an increasing number of samples.

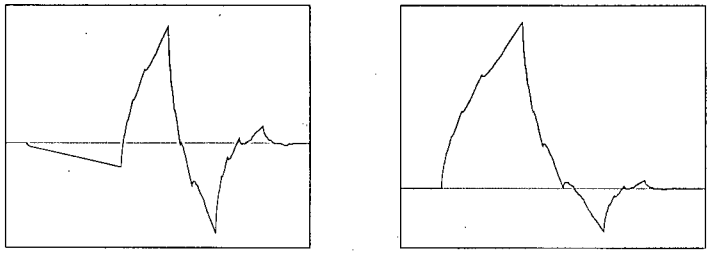

Figure A.2: Daubechies-4 Wavelet and Smoothing Functions. Daubechies wavelets were the first compact orthogonal wavelets discovered. This particular wavelet has two vanishing moments.

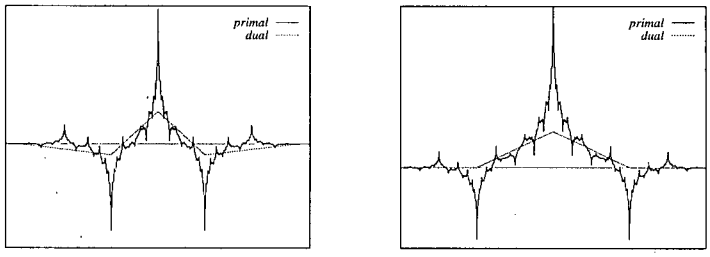

Figure A.3: Linear Spline Wavelet And Smoothing Functions. These wavelets are biorthogonal and have two vanishing moments. Unfortunately, while the dual wavelets and smoothing functions have a simple closed form, this is not true for the primals.

A.2 Multiresolution Refinement Equations

Since $\phi(x) \in V_0$ and $V_0 \subset V_1$, we can write $\phi(x)$ as a linear combination of the basis functions $\phi(2x - j)$ for V_1:

$$\phi(x) = \sqrt{2} \sum_j h_j \phi(2x - j) \tag{A.5}$$

This also holds for ψ:

$$\psi(x) = \sqrt{2} \sum_j g_j \phi(2x - j) \tag{A.6}$$

These are the *dilation* or *refinement equations*. Wavelet bases differ princi-

pally in their choices of $\{h_j\}$ (which determines $\{g_j\}$).

Using the enumerated bases:

$$\begin{aligned} \phi_{lm}(x) &= \sqrt{2}\sum_j h_j \phi_{l+1,2m+j}(x) \\ \psi_{lm}(x) &= \sqrt{2}\sum_j g_j \phi_{l+1,2m+j}(x) \end{aligned} \tag{A.7}$$

A.3 Orthogonal Wavelets

We define the inner product of two functions f and g with respect to a x:

$$\langle f \mid g \rangle_x \equiv \int_{-\infty}^{+\infty} f(x)g(x)dx \tag{A.8}$$

(Using x as a subscript is non-traditional, but will become useful when we consider multidimensional wavelets.)

Some wavelets are orthogonal:

$$\langle \psi_{lm} \mid \psi_{l'm'} \rangle_x = \delta_{ll'}\delta_{mm'} \tag{A.9}$$

But these have undesirable features: except for Haar, they are not symmetric and they do not include useful functions like splines.

A.4 Biorthogonal Wavelets

We can construct biorthogonal bases by using four functions instead of two: wavelets ϕ_{lm} and $\tilde{\phi}_{lm}$ and smoothing functions ψ_{lm} and $\tilde{\psi}_{lm}$. These are defined defined, respectively, by four sets of coefficients: $\{h_j\}$, $\{\tilde{h}_j\}$, $\{g_j\}$, and $\{\tilde{g}_j\}$. $\{h_j\}$ determines $\{\tilde{g}_j\}$ and $\{\tilde{h}_j\}$ determines $\{g_j\}$. If done consistently, the primal and dual components are entirely interchangeable.

For these,

$$\left\langle \psi_{lm} \mid \tilde{\psi}_{l'm'} \right\rangle_x = \delta_{ll'}\delta_{mm'} \tag{A.10}$$

In the rest of this section, we'll assume the more general biorthogonality, since we can always treat orthogonal wavelets as a special case of biorthogonal wavelets.

A.5 Wavelet Projections and Approximation

Let us discuss the ability of a wavelet representation to approximate an arbitrary function f. Let $\mathcal{P}_l f$ be the projection of a function $f \in L^2$ into the subspace V_l:

$$\mathcal{P}_l f(x) = \sum_m \left\langle f \mid \tilde{\phi}_{lm} \right\rangle_x \phi_{lm}(x) \tag{A.11}$$

It can be shown

$$\| f - \mathcal{P}_l f \|_2 \leq C 2^{-lN_v} \sqrt{\sum_n \| f^{(n)} \|^2} \tag{A.12}$$

where N_v is the number of vanishing moments of the wavelets, i. e. for $n = 0, \ldots, N_v - 1$

$$\int x^n \tilde{\psi}(x) dx = \left\langle x^n \mid \tilde{\psi}_{00} \right\rangle_x = 0 \ . \tag{A.13}$$

What Equation (A.12) means is that we can always decrease the L^2 error of a wavelet approximation by increasing the number of levels l or by choosing wavelets with a higher number of vanishing moments.

A.6 The Fast Wavelet Transform

The fast wavelet transform, first developed by Beylkin, et al. [5], starts with a set of data $s_{lm}, m = 0 \ldots N-1$, where $N = 2^l$. We treat these as coefficients of ϕ_{lm} and can compute a wavelet transform in $O(N)$ operations. Figure A.4 shows

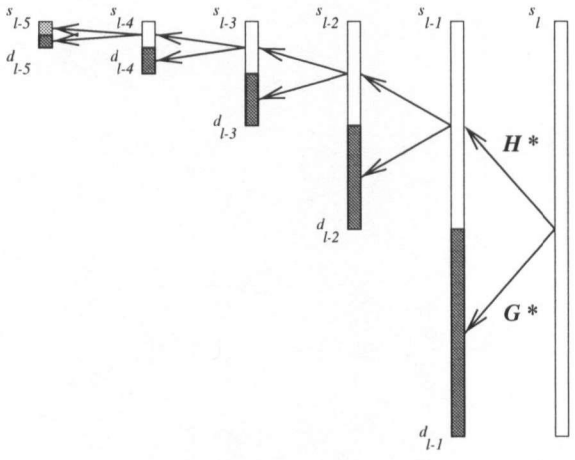

Figure A.4: The Fast Wavelet Transform

this schematically: starting on the right, each step operates on half the data of the previous step and there are l steps. This is another advantage of wavelet methods over Fourier techniques, which typically require $O(N \log N)$ operations.

The transform leaves us with $2^l - 1$ wavelet coefficients (dark gray in Figure A.4) and 1 smoothing coefficient (light gray). The hierarchical arrangement of these coefficients is sometimes referred to as the "wavelet pyramid".

A.7 Wavelet Compression

Even with the wavelet pyramid, we are still dealing with N coefficients. We have not saved any storage (yet).

It can be shown that if f is of the form

$$f(x) = \sum_{l,m} w_{lm} \psi_{lm}(x) \tag{A.14}$$

and we approximate it by

$$F(x) = \sum_{w_{lm} \in \tilde{W}} w_{lm} \psi_{lm}(x) \tag{A.15}$$

where the $\tilde{W}$ is a subset of the set of coefficients $\{w_{lm}\}$, that

$$\| f - F \|^2 = \sum_{w_{lm} \notin \tilde{W}} |w_{lm}|^2 \tag{A.16}$$

This gives us a convenient error metric. It also tells us that the optimal compression scheme discards the coefficients with smaller magnitudes first – a thresholding process.

Appendix B

Pseudocode for Haar Surface Interaction

Figures B.1-B.7 show the pseudocode for surface interaction with Haar wavelets. This implements Equations 5.47 and 7.1 with several optimizations, particularly "short circuiting": if one of the factors in Equation 5.47 is zero, the remaining factors are not computed.

```
# computing bR[N] = sum(J,K) f[J] bS[K] < < F[J] | B[K] > | B[N] >

# This attempt takes advantage of the sparsity of both f[] and bS[].

compute_bR:
   for all N
      bR[N] = 0
   for all J such that f[J] != 0
      for all K such that bS[K] != 0
         Delta_ks(f[J] * bS[K], J, K)                                    10
```

Figure B.1: Pseudocode for Surface Interaction I: top level

```
Delta_ks(s, J, K):
   if J.nu<0> == 0 && K.nu<2> == 0
      if K.l > J.l
         Q = 2^(K.l - J.l)
         if Q * J.m[0] <= K.m[2] < Q * (J.m[0] + 1)
            Delta_ls(s / sqrt(Q), J, K)
      else
         Q = 2^(J.l - K.l)
         if Q * K.m[2] <= J.m[0] < Q * (K.m[2] + 1)
            Delta_ls(s / sqrt(Q), J, K)                                  10
   else if J.nu<0> == 0 && K.nu<2> == 1
      if J.l > K.l
         Q = 2^(J.l - K.l)
         Qh = Q / 2
         if 0 <= J.m[0] - Q * K.m[2] < Qh
            Delta_ls(s / sqrt(Q), J, K)
         else if Qh <= J.m[0] - Q * K.m[2] < Q
            Delta_ls(-s / sqrt(Q), J, K)
   else if J.nu<0> == 1 && K.nu<2> == 0
      if J.l < K.l                                                       20
         Q = 2^(K.l - J.l)
         Qh = Q / 2
         if 0 <= K.m[0] - Q * J.m[2] < Qh
            Delta_ls(s / sqrt(Q), J, K)
         else if Qh <= K.m[0] - Q * J.m[2] < Q
            Delta_ls(-s / sqrt(Q), J, K)
   else if J.l == K.l && K.m[2] == J.m[0]
      Delta_ls(s, J, K)
```

Figure B.2: Pseudocode for Surface Interaction II: Delta_ks

```
Delta_ls(s, J, K):
   if J.nu<1> == 0 && K.nu<3> == 0
      if K.l > J.l
         Q = 2^(K.l - J.l)
         if 0 <= K.m[3] - Q * J.m[1] < Q
            Delta_kr(s / sqrt(Q), J, K)
      else
         Q = 2^(J.l - K.l)
         if 0 <= J.m[1] - Q * K.m[3] < Q
            Delta_kr(s / sqrt(Q), J, K)                          10
   else if J.nu<1> == 0 && K.nu<3> == 1
      if J.l > K.l
         Q = 2^(J.l - K.l)
         Qh = Q / 2
         if 0 <= J.m[1] - Q * K.m[3] < Qh
            Delta_kr(s / sqrt(Q), J, K)
         else if Qh <= J.m[1] - Q * K.m[3] < Q
            Delta_kr(-s / sqrt(Q), J, K)
   else if J.nu<1> == 1 && K.nu<3> == 0
      if J.l < K.l                                               20
         Q = 2^(K.l - J.l)
         Qh = Q / 2
         if 0 <= J.m[1] - Q * K.m[3] < Qh
            Delta_kr(s / sqrt(Q), J, K)
         else if Qh <= J.m[1] - Q * K.m[3] < Q
            Delta_kr(-s / sqrt(Q), J, K)
   else if K.l == J.l && K.m[3] == J.m[1]
      Delta_kr(s, J, K)
```

Figure B.3: Pseudocode for Surface Interaction III: Delta_ls

```
Delta_kr(s, J, K):
   for N.nu = 0 to 15
      if J.nu<2> == 0 && N.nu<2> == 0
         lNmax = ( N.nu == 0 ? 0 : maximum possible level )
         for N.l = 0 to lNmax
            if N.l > J.l
               Q = 2^(N.l - J.l)
               for delM = 0 to Q - 1
                  N.m[2] = Q * J.m[2] + delM
                  Delta_lr(s / sqrt(Q), J, K, N.nu, N.l, N.m[2])
            else
               Q = 2^(J.l - N.l)
               N.m[2] = floor(J.m[2] / Q)
               Delta_lr(s / sqrt(Q), J, K, N.nu, N.l, N.m[2])
      else if J.nu<2> == 0 && N.nu<2> == 1
         lNmax = ( N.nu == 0 ? 0 : J.l - 1 )
         for N.l = 0 to lNmax
            Q = 2^(J.l - N.l)
            Qh = Q / 2
            N.m[2] = floor(J.m[2] / Q)
            if 0 <= J.m[2] - Q * N.m[2] < Qh
               Delta_lr(s / sqrt(Q), J, K, N.nu, N.l, N.m[2])
            else
               Delta_lr(-s / sqrt(Q), J, K, N.nu, N.l, N.m[2])
      else if J.nu<2> == 1 && N.nu<2> == 0
         lNmax = ( N.nu == 0 ? 0 : maximum possible level )
         for N.l = J.l + 1 to lNmax
            Q = 2^(J.l - N.l)
            Qh = Q / 2
            for delM = 0 to Qh - 1
               N.m[2] = Q * J.m[2] + delM
               Delta_lr(s / sqrt(Q), J, K, N.nu, N.l, N.m[2])
            for delM = Qh to Q - 1
               N.m[2] = Q * J.m[2] + delM
               Delta_lr(-s / sqrt(Q), J, K, N.nu, N.l, N.m[2])
      else if N.nu != 0 || J.l == 0
         N.l = J.l
         N.m[2] = J.m[2]
         Delta_lr(s, J, K, N.nu, N.l, N.m[2])
```

Figure B.4: Pseudocode for Surface Interaction IV: Delta_kr

```
Delta_lr(s, J, K, N.nu, N.l, N.m[2]):
   if J.nu<3> == 0 && N.nu<3> == 0
      if N.l > J.l
         Q = 2^(N.l - J.l)
         for delM = 0 to Q - 1
            N.m[3] = Q * J.m[3] + delM
            Delta_x(s / sqrt(Q), J, K, N.nu, N.l, N.m[2], N.m[3])
      else
         Q = 2^(J.l - N.l)
         N.m[3] = floor(J.m[3] / Q)                                    10
         Delta_x(s / sqrt(Q), J, K, N.nu, N.l, N.m[2], N.m[3])
   else if J.nu<3> == 0 && N.nu<3> == 1
      if J.l > N.l
         Q = 2^(J.l - N.l)
         Qh = Q / 2
         N.m[3] = floor(J.m[3] / Q)
         if 0 <= J.m[3] - Q * N.m[3] < Qh
            Delta_x(s / sqrt(Q), J, K, N.nu, N.l, N.m[2], N.m[3])
         else
            Delta_x(-s / sqrt(Q), J, K, N.nu, N.l, N.m[2], N.m[3])     20
   else if J.nu<3> == 1 && N.nu<3> == 0
      if J.l < N.l
         Q = 2^(N.l - J.l)
         Qh = Q / 2
         for delM = 0 to Qh - 1
            N.m[3] = Q * J.m[3] + delM
            Delta_x(s / sqrt(Q), J, K, N.nu, N.l, N.m[2], N.m[3])
         for delM = Qh to Q - 1
            N.m[3] = Q * J.m[3] + delM
            Delta_x(-s / sqrt(Q), J, K, N.nu, N.l, N.m[2], N.m[3])     30
   else if N.l == J.l
      N.m[3] = J.m[3]
      Delta_x(s, J, K, N.nu, N.l, N.m[2], N.m[3])
```

Figure B.5: Pseudocode for Surface Interaction V: Delta_lr

```
Delta_x(s, J, K, N.nu, N.l, N.m[2]):
  if K.nu<0> == 0 && N.nu<0> == 0
    if N.l > K.l
      Q = 2^(N.l - K.l)
      for delM = 0 to Q - 1
        N.m[0] = Q * K.m[0] + delM
        Delta_y(s / sqrt(Q), J, K, N.nu, N.l, N.m[0], N.m[2], N.m[3])
    else
      Q = 2^(K.l - N.l)
      N.m[0] = floor(K.m[0] / Q)                                          10
      Delta_y(s / sqrt(Q), J, K, N.nu, N.l, N.m[0], N.m[2], N.m[3])
  else if K.nu<0> == 0 && N.nu<0> == 1
    if K.l > N.l
      Q = 2^(K.l - N.l)
      Qh = Q / 2
      N.m[0] = floor(K.m[0] / Q)
      if 0 <= K.m[0] - Q * N.m[0] < Qh
        Delta_y(s / sqrt(Q), J, K, N.nu, N.l, N.m[0], N.m[2], N.m[3])
      else
        Delta_y(-s / sqrt(Q), J, K, N.nu, N.l, N.m[0], N.m[2], N.m[3])    20
  else if K.nu<0> == 1 && N.nu<0> == 0
    if K.l < N.l
      Q = 2^(N.l - K.l)
      Qh = Q / 2
      for delM = 0 to Qh - 1
        N.m[0] = Q * K.m[0] + delM
        Delta_y(s / sqrt(Q), J, K, N.nu, N.l, N.m[0], N.m[2], N.m[3])
      for delM = Qh to Q - 1
        N.m[0] = Q * K.m[0] + delM
        Delta_y(s / sqrt(Q), J, K, N.nu, N.l, N.m[0], N.m[2], N.m[3])     30
  else if K.l == N.l
    N.m[0] = K.m[0]
    Delta_y(s, J, K, N.nu, N.l, N.m[0], N.m[2], N.m[3])
```

Figure B.6: Pseudocode for Surface Interaction VI: Delta_x

```
Delta_y(s, J, K, N.nu, N.l, N.m[0], N.m[2], N.m[3]):
   if K.nu<1> == 0 && N.nu<1> == 0
      if N.l > K.l
         Q = 2^(N.l - K.l)
         for delM = 0 to Q - 1
            N.m[1] = Q * K.m[1] + delM
            bR[N] = bR[N] + s / sqrt(Q)
      else
         Q = 2^(K.l - N.l)
         N.m[1] = floor(K.m[1] / Q)                                    10
         bR[N] = bR[N] + s / sqrt(Q)
   else if K.nu<1> == 0 && N.nu<1> == 1
      if K.l > N.l
         Q = 2^(K.l - N.l)
         Qh = Q / 2
         N.m[1] = floor(K.m[1] / Q)
         if 0  <= K.m[1] - Q * N.m[1] < Qh
            bR[N] = bR[N] + s / sqrt(Q)
         else
            bR[N] = bR[N] - s / sqrt(Q)                                20
   else if K.nu<1> == 1 && N.nu<1> == 0
      if K.l < N.l
         Q = 2^(N.l - K.l)
         Qh = Q / 2
         for delM = 0 to Qh - 1
            N.m[1] = Q * K.m[1] + delM
            bR[N] = bR[N] + s / sqrt(Q)
         for delM = Qh to Q - 1
            N.m[1] = Q * K.m[1] + delM
            bR[N] = bR[N] - s / sqrt(Q)                                30
   else if K.l == N.l
      N.m[1] = K.m[1]
      bR[N] = bR[N] + s
```

Figure B.7: Pseudocode for Surface Interaction VII: Delta_y

Bibliography

[1] James Arvo and David Kirk. A survey of ray tracing acceleration techniques. In Andrew S. Glassner, editor, *An Introduction to Ray Tracing*, pages 201–262. Academic Press, 1989.

[2] Ian Ashdown. Near-field photometry: The helios approach. In *Graphics Interface '92 Workshop on Local Illumination*, pages 1–14, May 1992.

[3] Larry Aupperle and Pat Hanrahan. Importance and discrete three point transport. In Michael F. Cohen, Claude Puech, and Francois Sillion, editors, *Fourth Eurographics Workshop on Rendering*, pages 85–94. Eurographics, June 1993. held in Paris, France, 14–16 June 1993.

[4] P. Beckmann and A. Spizzichino. *The scattering of electromagnetic waves from rough surf aces.* Artech House Inc, 1987.

[5] G. Beylkin, Ronald R. Coifman, and V. Rokhlin. Fast wavelet transforms and numerical algorithms i. *Communications on Pure and Applied Mathematics*, 44:141–183, 1991.

[6] James F. Blinn. Models of light reflection for computer synthesized pictures. In James George, editor, *Computer Graphics (SIGGRAPH '77 Proceedings)*, volume 11, pages 192–198, San Jose, California, July 1977.

[7] James F. Blinn. Simulation of wrinkled surfaces. In *Computer Graphics (SIGGRAPH '78 Proceedings)*, volume 12, pages 286–292, August 1978.

[8] OpenGL Architecture Review Board, Mason Woo, Jackie Neider, and Tom Davis. *OpenGL Programming Guide.* Addison Wesley Developers Press, 1997.

[9] Edwin E. Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces.* PhD thesis, Dept. of CS, U. of Utah, December 1974.

[10] S. Chandrasekar. *Radiative Transfer.* Oxford Univ. Press, 1950.

[11] Per H. Christensen, Eric J. Stollnitz, David H. Salesin, and Tony D. DeRose. Wavelet radiance. In *Fifth Eurographics Workshop on Rendering*, pages 287–302, Darmstadt, Germany, June 1994.

[12] Per Henrik Christensen, Eric J. Stollnitz, David H. Salesin, and Tony D. DeRose. Global illumination of glossy environments using wavelets and importance. *ACM Transactions on Graphics*, 15(1):37–71, January 1996.

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

[14] Michael F. Cohen and John R. Wallace. *Radiosity and Realistic Image Synthesis.* Academic Press Professional, San Diego, CA, 1993.

[15] R. L. Cook and K. E. Torrance. A reflectance model for computer graphics. *ACM Transactions on Graphics*, 1(1):7–24, January 1982.

[16] Ingrid Daubechies. *Ten Lectures on Wavelets*, volume 61 of *CBMS-NSF Regional Conference Series in Applied Mathematics.* SIAM, Philadelphia, PA, 1992.

[17] Eugene L. Fiume. *The Mathematical Structure of Raster Graphics.* Academic Press, April 1989.

[18] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics, Principles and Practice, Second Edition.* Addison-Wesley, Reading, Massachusetts, 1990. Overview of research to date.

[19] A. Fournier. Separating reflection functions for linear radiosity. In *Eurographics Rendering Workshop 1995.* Eurographics, June 1995.

[20] A. Fournier, E. Fiume, M. Ouellette, and C. K. Chee. FIAT LUX: Light driven global illumination. Technical Memo DGP89–1, Dynamic Graphics Project, Department of Computer Science, University of Toronto, 1989.

[21] Alain Fournier. Normal distribution functions and multiple surfaces. In *Graphics Interface '92 Workshop on Local Illumination*, pages 45–52, May 1992.

[22] Alain Fournier. From local to global illumination and back. In P. M. Hanrahan and W. Purgathofer, editors, *Rendering Techniques '95 (Proceedings of the Sixth Eurographics Workshop on Rendering)*, pages 127–136, Dublin, Ireland, June 1995. Springer-Verlag.

[23] Akira Fujimoto, Takayuki Tanaka, and Kansei Iwata. Arts: Accelerated ray-tracing system. *IEEE Computer Graphics and Applications*, pages 16–26, April 1986.

[24] Andrew S. Glassner. *Principles of Digital Image Synthesis.* Morgan Kaufmann, 1995.

[25] Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile. Modelling the interaction of light between diffuse surfaces. In *Computer Graphics (SIGGRAPH '84 Proceedings)*, volume 18, pages 212–22, July 1984.

[26] Steven J. Gortler, Michael F. Cohen, and Phillip Slusallek. Radiosity and relaxation methods: Progressive refinement is southwell relaxation. Technical Report CS-TR-408-93, Department of Computer Science, Princeton University, February 1993. to appear in IEEE Computer Graphics & Applications.

[27] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. In Holly Rushmeier, editor, *Proceedings of SIGGRAPH '96 (New Orleans, LA, August 4-9, 1996)*, Computer Graphics Proceedings, Annual Conference Series, pages 43–54. ACM SIGGRAPH, ACM Press, August 1996.

[28] Steven J. Gortler, Peter Schroder, Michael F. Cohen, and Pat Hanrahan. Wavelet radiosity. In *Computer Graphics Proceedings, Annual Conference Series, 1993*, pages 221–230, 1993.

[29] Eric A. Haines. A proposal for standard graphics environments. *IEEE Computer Graphics and Applications*, 7(11):3–5, November 1987.

[30] Roy Hall. *Illumination and Color in Computer Generated Imagery.* Springer-Verlag, New York, 1989.

[31] Pat Hanrahan and Wolfgang Krueger. Reflection from layered surfaces due to subsurface scattering. In James T. Kajiya, editor, *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 165–174, August 1993.

[32] Pat Hanrahan, David Salzman, and Larry Aupperle. A rapid hierarchical radiosity algorithm. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 197–206, July 1991.

[33] Xiao D. He, Kenneth E. Torrance, Francois X. Sillion, and Donald P. Greenberg. A comprehensive physical model for light reflection. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 175–186, July 1991.

[34] Berthold Klaus Paul Horn. *Robot Vision.* McGraw-Hill, 1986.

[35] David S. Immel, Michael F. Cohen, and Donald P. Greenberg. A radiosity method for non-diffuse environments. In David C. Evans and Russell J. Athay, editors, *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20, pages 133–142, August 1986.

[36] American National Standards Institute. ANSI standard nomenclature and definitions for illuminating engineering,. ANSI/IES RP-16-1986, Illuminating Engineering Society, 345 East 47th Street, New York, NY 10017, June 1986.

[37] Henrik Wann Jensen. Global illumination using photon maps. In *Seventh Eurographics Workshop on Rendering*, pages 22–30, Porto, Portugal, June 1996.

[38] James T. Kajiya. The rendering equation. In David C. Evans and Russell J. Athay, editors, *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20, pages 143–150, August 1986.

[39] Alexander Keller. Quasi-monte carlo radiosity. In *Seventh Eurographics Workshop on Rendering*, Porto, Portugal, June 1996.

[40] Renate Kempf and Chris Frazier, editors. *OpenGL Reference Manual.* Addison Wesley Developers Press, 1997.

[41] Paul Lalonde and Alain Fournier. Filtered local shading in the wavelet domain. In J. Dorsey and Ph. Slusallek, editors, *Rendering Techniques '97 (Proceedings of the 8th Eurographics Workshop on Rendering)*, pages 163–174. Springer-Verlag, June 1997.

[42] Paul Lalonde and Alain Fournier. A wavelet representation of reflectance functions. *IEEE Transactions on Visualization and Computer Graphics*, 3(4):329–336, October–December 1997.

[43] J. Lane, L. Carpenter, T. Whitted, and J. Blinn. Scan line methods for displaying parametrically defined surfaces. *Communications of the ACM*, 23(1):23–34, 1980.

[44] Marc Levoy and Pat Hanrahan. Light field rendering. In Holly Rushmeier, editor, *Proceedings of SIGGRAPH '96 (New Orleans, LA, August 4-9, 1996)*, Computer Graphics Proceedings, Annual Conference Series, pages 31–42. ACM SIGGRAPH, ACM Press, August 1996.

[45] Robert Lewis. Making shaders more physically plausible. In Michael F. Cohen, Claude Puech, and Francois Sillion, editors, *Fourth Eurographics Workshop on Rendering*, pages 47–62. Eurographics, June 1993. held in Paris, France, 14–16 June 1993.

[46] Robert R. Lewis. Solving the classic radiosity equation using multigrid techniques. In *Proceedings of the 1992 Western Computer Graphics Symposium*, pages 157–164, Banff, Alberta, Canada, April 1992.

[47] Robert R. Lewis. Making shaders more physically plausible. *Computer Graphics Forum*, 13(2):109 – 120, June 1994.

[48] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In Maureen C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 163–169, July 1987.

[49] Martti J. Mantyla. *Introduction to Solid Modeling.* Computer Science Press, Rockville, MD, December 1986.

[50] Steve McConnell. *Code Complete.* Microsoft Press, 1993.

[51] Dimitri Mihalas. *Stellar Atmospheres.* W. H. Freeman, San Francisco, 1978.

[52] M. Minnaert. The reciprocity principle in lunar photometry. *Astrophysical Journal*, 93:403–410, May 1941.

[53] Curtis D. Mobley. *Light in Water: Radiative Transfer in Natural Waters.* Academic Press, San Diego, 1994.

[54] Hans Moravec. 3D graphics and the wave theory. In *Computer Graphics (SIGGRAPH '81 Proceedings)*, volume 15, pages 289–96, August 1981.

[55] L. Neumann and A. Neumann. Photosimulation: Interreflection with arbitrary reflectance models and illuminations. *Computer Graphics Forum*, 8(1):21–34, March 1989.

[56] W. Nusselt. Graphische Bestimmung des Winkelverhältnisses bei der Wärme-Strahlung. *Zeitschrift des Vereines Deutscher Ingenieure*, 72(20):673, 1928.

[57] Michael Oren and Shree K. Nayar. Generalization of Lambert's reflectance model. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 239–246. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

[58] Bui-T. Phong. Illumination for computer generated pictures. *Communications of the ACM*, 18(6):311–317, June 1975.

[59] Pierre Poulin and John Amanatides. Shading and shadowing with linear light sources. In C. E. Vandoni and D. A. Duce, editors, *Eurographics '90*, pages 377–386. North-Holland, September 1990.

[60] Pierre Poulin and Alain Fournier. A model for anisotropic reflection. In Forest Baskett, editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 273–282, August 1990.

[61] Mark C. Reichert. A two-pass radiosity method driven by lights and viewer position. Master's thesis, Program of Computer Graphics, Cornell University, January 1992.

[62] Lawrence G. Roberts. Machine perception of three-dimensional solids. TR 315, Lincoln Lab, MIT, Lexington, MA, May 1963.

[63] Holly Rushmeier. Rendering participating media: Problems and solutions from application areas. In *Fifth Eurographics Workshop on Rendering*, pages 35–56, Darmstadt, Germany, June 1994.

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

[65] Peter Schroeder, Steven Gortler, Michael Cohen, and Pat Hanrahan. Wavelet projections for radiosity. In Michael F. Cohen, Claude Puech, and Francois Sillion, editors, *Fourth Eurographics Workshop on Rendering*, pages 105–114. Eurographics, June 1993. held in Paris, France, 14–16 June 1993.

[66] Peter Schroeder and Pat Hanrahan. Wavelet methods for radiance computations. In *Fifth Eurographics Workshop on Rendering*, pages 303–311, Darmstadt, Germany, June 1994.

[67] Michael Shantz and Sheue-Ling Lien. Shading bicubic patches. In Maureen C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 189–196, July 1987.

[68] Robert Siegel and John R. Howell. *Thermal Radiation Heat Transfer*. Hemisphere Publishing Corp., Washington, DC, 1981.

[69] Francois X. Sillion, James R. Arvo, Stephen H. Westin, and Donald P. Greenberg. A global illumination solution for general reflectance distributions. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 187–196, July 1991.

[70] Charles Simonyi. Meta-programming: A software production method. Technical Report CSL-76-7, Xerox Palo Alto Research Center, 1976.

[71] Brian Smits, James Arvo, and Donald Greenberg. A clustering algorithm for radiosity in complex environments. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 435–442. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

[72] Brian E. Smits, James R. Arvo, and David H. Salesin. An importance-driven radiosity algorithm. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 273–282, July 1992.

[73] Barton T. Stander and John C. Hard. Guaranteeing the topology of an implicit surface polygonalization for interactive modeling. In *SIGGRAPH '97 Conference Proceedings*, pages 279–286, Los Angeles, California, August 3-8 1997.

[74] Toshimitsu Tanaka and Tokiichiro Takahashi. Shading with area light sources. *Eurographics '91*, pages 235–46, 535–7, September 1991.

[75] K. E. Torrance and E. M. Sparrow. Theory for off-specular reflection from roughened surfaces. *Journal of Optical Society of America*, 57(9), 1967.

[76] T. S. Trowbridge and K. P. Reitz. Average irregularity representation of a roughened surfaces for ray reflection. *Journal of Optical Society of America*, 65(3), 1967.

[77] C. P. Verbeck and D. P. Greenberg. A comprehensive light source description for computer graphics. *IEEE Computer Graphics and Applications*, 4(7):66–75, July 1984.

[78] Gregory J. Ward. Measuring and modeling anisotropic reflection. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 265–272, July 1992.

[79] J. Warnock. A hidden-surface algorithm for computer generated half-tone pictures. Technical Report TR 4–15, NTIS AD-733 671, University of Utah, Computer Science Department, 1969.

[80] Alan Watt. *Fundamentals of Three-Dimensional Computer Graphics.* Addison-Wesley, Wokingham, England, 1989.

[81] K. Weiler and K. Atherton. Hidden surface removal using polygon area sorting. *Computer Graphics (SIGGRAPH '77 Proceedings)*, 11(2):214–222, July 1977.

[82] Stephen H. Westin, James R. Arvo, and Kenneth E. Torrance. Predicting reflectance functions from complex surfaces. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 255–264, July 1992.

[83] Turner Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 23(6):343–349, June 1980.

[84] Lance Williams. Casting curved shadows on curved surfaces. In *Computer Graphics (SIGGRAPH '78 Proceedings)*, volume 12, pages 270–274, August 1978.

[85] R. J. Woodham and T. K. Lee. Photometric method for radiometric correction of multispectral scanner data. *Canadian Journal of Remote Sensing*, 11(2):132–161, December 1985.

[86] Brian Wyvill, Craig McPheeters, and Geoff Wyvill. Data structure for soft objects. *The Visual Computer*, 2(4):227–234, 1986.

[87] Daniel Zwillinger. *Handbook of Integration.* Jones and Bartlett Publishers, Inc., Boston, MA, 1992.

Index