Analysis of Radiosity Techniques in Computer Graphics

Bernard Kwok

A thesis submitted in conformity with the requirements
for the degree of Master of Science

Department of Computer Science
York University

May 1992

Acknowledgments

Special thanks goes to my thesis advisor Professor John Amanatides, for guidance in my thesis work, and for providing access to numerous reference material. Thanks also to Professor Michael Jenkin for help in implementation issues on the graphics workstations, and various lab equipment. Thanks to Dan Baum at Silicon Graphics Inc., Pat Hanrahan at Princeton University, and John Wallace at 3D/Eye Inc. for insights into their implementations, and preprints of their papers. Finally, thanks to Holly Rushmeier at Georgia Tech., Francois Sillion at Ecole Normal Superiere, and Eric Chen at Apple Computer for access to their papers and preprints.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

In the past twenty years or so, computer graphics techniques for simulating the interaction of light with matter has progressed to the point where photorealistic quality images can be produced quite efficiently. While earlier algorithms considered direct lighting only, solutions to the global illumination problem with indirect lighting, surface to surface interreflections, and shadows are now tractable. The two major algorithmic approaches that have arisen to solve this problem are ray tracing and radiosity. This thesis will examine the radiosity method and provide an implementation of important advances in this method.

The current state of workstation graphics hardware and software offers rendering of objects that are directly illuminated by light sources. Usually objects can have surfaces that have diffuse or specular properties, and polygons may be rendered using flat shading, linearly interpolated shading [Gour71] or with highlighting [Phong75]. Though large numbers of surfaces can be rendered at high speeds (roughly 25,000 polygons per second), the resulting images can still be easily recognized as computer generated simulations. The major problem is the fact that these renderings do not consider the effect of interactions between surfaces in the environment, that is, indirect light. In the real world, surfaces are lit both *directly* by light sources and *indirectly* from reflections off other surfaces. Even though these effects are difficult to simulate, they are of great importance for realistic image synthesis.

Realistic simulations depend to a large extent on the global reflection model at each surface in the environment. This model defines the outgoing intensities in any direction as a function of incoming energy, (the global illumination), from all directions.

In recent years two methods, ray tracing and radiosity, have been able to simulate global illumination effects in a tractable manner, but both are still restrictive. Ray tracing has become the preferred method for environments that have predominantly specular surfaces, using discrete sampling methods, to give results that are dependent on the position from which the environment is viewed (*view dependent*). The radiosity method, performs well for diffuse scenes, using discrete models of the environment, and giving results that are *view independent*.

The introduction of the radiosity method has led to a loosening of a number of image synthesis constraints. The method provides a separation of the simulation of light inter-reflection, and rendering of the final image. This has allowed for computation of the illumination once, with different views of the environment only requiring recomputation of the rendering step. Standard graphics hardware pipelines on many of todays graphics workstations allow for dynamic sequences of these images to be displayed. In addition, more accurate solutions can be achieved giving effects such as "colour bleeding", complex shading / shadow effects, and simulation of area light sources.

The radiosity method has since evolved to allow for more physical accuracy, faster computation and less storage cost, incorporation of more complex physical models, and increased usability through pre (model generation) and post (rendering) processes.

1.2 The Radiosity Method

The basic radiosity method is based on principles from the field of thermal engineering. The underlying theory can be found in most radiative heat transfer texts ([Siegel81] [Sparrow63] [Sparrow78]). In general we are dealing with a number of surfaces which form an enclosure. Each of these surfaces, and the volume enclosed have both radiative and geometric properties associated with them. The aim of the radiosity method is to compute the radiosity for every surface, where the *radiosity* is basically the energy per unit time, per unit area leaving a surface due to the original energy of the surface plus any impinging energy that is reflected off of it. In order to compute impinging energy, geometrical relations called *form-factors* need to be computed between pairs of surfaces, for surface-volume, or for volume-volume interactions to determine the amount of energy that arrives at a surface from other surfaces or volumes, either directly or indirectly. For surface-surface interactions, form-factors are based on the shape, area, and orientation of each surface, the distance between them, and the portion of each surface visible to the other. If a volume is involved, then attenuation or enhancement of radiosity transport due to volumetric properties must also be considered.

The assumption that the enclosure encloses a vacuum, and thus volume interactions are not involved, has been used by most of the current research into determining solutions. For this case the fundamental equation for computing surface radiosity is:

$$Radiosity_i = Emission_i + \sum_j Reflectance_{ij} \left(Radiosity_j \times FormFactor_{i-j} \right) \tag{1.1}$$

where single subscript i denotes surface i, and ij denotes i with respect to j. The equation basically says that the radiosity of some surface i is equal to the original emission of that surface plus the sum of any reflected impinging energy from every other surface j in the environment (See *Figure 1.1a*). The impinging energy is dependent upon the radiosity of all surfaces j ($Radiosity_j$) and the form-factor ($FormFactor_{i-j}$) which determines the fraction of the radiosity of surface j that arrives at surface i.

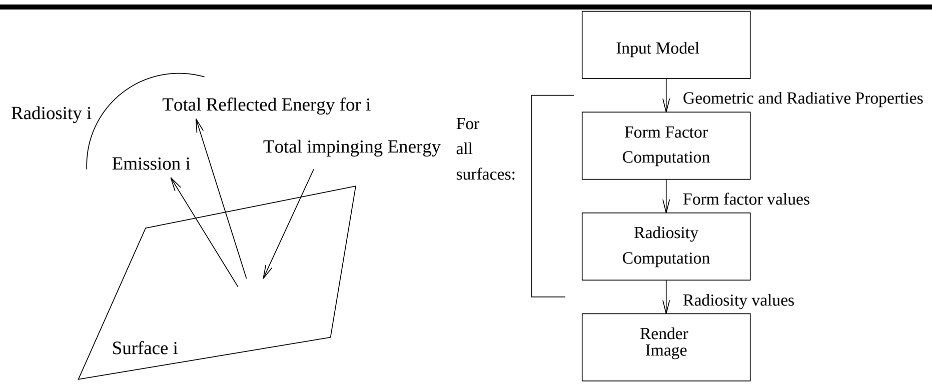

Figure 1.1: a) Geometry for radiosity of a surface (i) b) Flowchart of basic radiosity method.

Examining equation (1.1), note that the radiosity of a surface i is dependent on the radiosity of every other surface j resulting in a system of linear equations of the form of equation (1.1) being required, with one equation for each of the surfaces. The task of calculating the global illumination is thus reduced to solving the form-factors between every pair of surfaces, and then solving the resulting set of linear equations to find the radiosity values for all surfaces i. Once computed, images may be rendered. A basic outline of the method is given in *Figure 1.1b*.

1.3 Motivation

Currently there are few articles that go beyond a simple survey of one or more areas involved in the radiosity method ([Cohen86], [Pueyo91]) to perform a comparative examination of all areas to any great depth. Our first aim is to help to fill this void. The major areas of research covered in our survey includes modeling, form-factor computation, radiosity computation, rendering, and parallelization. Since the most computationally expensive part of the process is computing form-factors, it has received the most research attention. Much of the remaining research has gone into computing solutions with less restrictive illumination models. As such, a major part of the survey concentrates on these two areas.

In order to examine these areas more closely, an implementation of some of the more important advances in the radiosity method is given. There are currently a number of publicly available implementations using the ray tracing method (e.g. **RayTrace** [Kolb], **Radiance** [Watt], and **DKBTrace** [Buck]). Part of our motivation is to provide a publicly available general implementation using the radiosity method, which is currently lacking. We also hope to provide some comparative experimental results using previous separately tested techniques in conjunction with one another.

For continuity, our discussion will cover the main constituents of a radiosity method in a step-by-step manner, starting with initial model creation, then simulation to compute radiosity values, and finally rendering of images. For each part of the process, issues such as computational cost, use of coherence, assumptions, compatibility, and physical accuracy will be addressed.

1.4 Outline of Thesis

This thesis is composed of three main parts. The first part will provide background for those unfamiliar with the heat transfer principles that underly the radiosity method. The second part will contain a careful examination of various issues related to important areas of research into the radiosity method. The third will be an implementation of some important results. Chapters 2 and 3 contain heat transfer background. Chapter 4 contains our survey, Chapter 5 our radiosity implementation, and Chapter 6 initial experimental results comparing the various techniques used. Chapter 7 provides some closing remarks, looking at outstanding issues, and future directions. Appendix A contains experimental statistics, Appendix B a glossary of some common terms used in this thesis, and Appendix C an example of using techniques given in Chapter 4.

Chapter 2

Thermal Engineering Concepts

2.1 Introduction

Various heat transfer principles within the field of thermal engineering have been used as the basis of the radiosity method in computer graphics. In this chapter some important material will be summarized from this area. This material can be found in most radiative heat transfer texts ([Sarofim67], [Sparrow78], [Sparrow68], [Siegel81], [Chapman89]).

2.2 Radiation Heat Transfer

There are basically three modes of heat transfer between objects: conduction, convection, and radiation. We are interested in *radiation*. The basic idea behind radiation is that when part or all of body is hotter than it's surroundings it tends to cool with time, by emitting energy in all directions. Part of this energy may be reflected, transmitted, and / or absorbed.

More succinctly, electromagnetic radiation is observed to be emitted at the surface of a body which has been thermally excited, such that the following rules hold:

- There is usually an exchange of energy from a hotter to a cooler body.
- The net exchange of energy is zero.
- The exchange is size, shape, and relative orientation dependent.

If the energy exchange can occur without a medium of transport present, then the mechanism involved is *thermal radiation*.

In the sections to follow, thermal radiation energy transport will be discussed in some detail. The first part of the discussion will explain some basic principles and terminology. Our focus will then be on the direct exchange of energy between two bodies using the concept of a *form-factor*. Lastly, the total exchange of energy in environments with certain radiative characteristics will be considered.

2.3 Geometric, Surface and Radiative Properties

This section will outline some basic definitions and relations for various geometric, surface, and radiative properties, that will be needed later on.

Before starting, some terms that are used to categorize surface and radiative properties are given:

- If a radiative or surface property is wavelength dependent then there is a *spectral dependency*. A *monochromatic* property is one that is applicable at one wavelength, A *total* property is one that is evaluated over the entire applicable wavelength spectrum. Monochromatic properties will be denoted by subscript λ.
- A property that has *directional dependencies*, may be applicable only in a single direction, or be a value summed over all directions above a surface (*hemispherical*). If applicable to a single direction, a property will be accompanied by some directional coordinates.

2.3.1 Basic Definitions

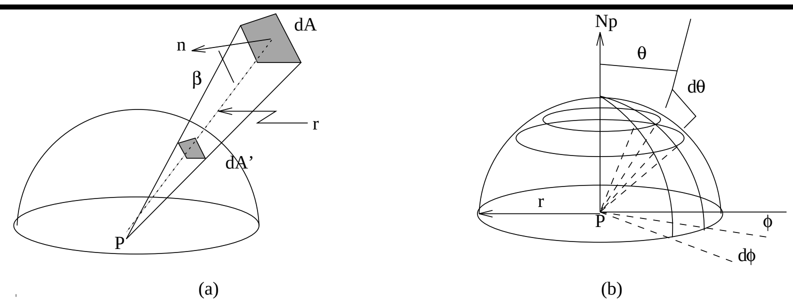

Figure 2.1: Differential solid angle geometry: a) In terms of differential projected area, and b) In terms of polar coordinates of a hemisphere.

Terms

- A *solid angle* (ω) is the angle of the apex of a cone, and is measured in terms of the area on a hemisphere intercepted by a cone whose apex is at the hemispheres center P. A solid angle is measured in terms of a *steradian* (sr) which is the solid angle of a cone that intercepts an area equal to the square of the hemisphere's radius r. There are 2π steradian per hemisphere.

  An alternative way to view a solid angle is as follows: The *solid angle subtended by a surface*, is the area (A) of a surface projected down onto the hemisphere centered at some point P, divided by the square of the hemisphere's radius. Thus, the differential solid angle subtended by a differential area dA located distance r from P is:

$$d\omega = \frac{dA'}{r^2} = \frac{cos\beta dA}{r^2} \tag{2.1}$$

  where dA' is the projection of dA normal to r and β is the angle between r and normal drawn to dA.

  In terms of polar angle θ and azimuth angle ϕ (longitude), a differential solid angle is given as:

$$d\omega = \frac{dA'}{r^2} = \frac{(rd\theta)(rsin\theta d\phi)}{r^2} = sin\theta d\theta d\phi \tag{2.2}$$

  (See *Figure 2.1*).

- *energy flux* (q) is the general term for the rate at which energy is emitted

- *energy flux density* (dq) is the energy leaving a surface per unit time and unit area, contained within a differential solid angle ($d\omega$), in a given direction (θ) measured from the surface normal to the axis of the solid angle.

$$dq = I \cos\theta d\omega \tag{2.3}$$

- *Radiance* (I) or *intensity of radiation* is the directional distribution of energy to or from a surface (by emission or reflection) per unit area normal to the solid angle, per differential solid angle, per unit time. From the definition for dq (Equation 2.3), I is given as:

$$I = \frac{dq}{\cos\theta d\omega} \tag{2.4}$$

- *Emissive power* (E) is the emitted thermal energy leaving a surface, per unit time, per unit area of a surface, and may be spectrally and directionally dependent. It is original emission from a surface, not the reflected energy. E is dependent on the temperature, substance, and structure (e.g. roughness) of the emitting surface,

- *Radiosity* (B) is the radiation leaving a surface per unit time and unit area. This value includes both reflected energy and original emission.

- *Irradiation* (H) is the rate per unit area, at which thermal energy is incident on a surface (from all directions). Irradiation results from emissions and reflections from all other surfaces, and may be spectrally dependent.

- *Reflectivity* (ρ) gives the amount of the radiation incident to the surface of a body that is reflected by the body.

- *Absorptivity* (α) gives the amount of the radiation incident to the surface of a body that is absorbed by the body.
- *Transmissivity* (τ) gives the amount of the radiation incident to the surface of a body that is transmitted through the body.

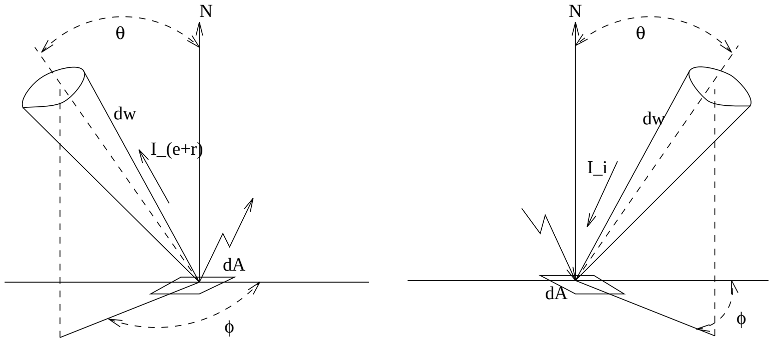

Figure 2.2: a) Radiosity geometry b) Irradiation geometry

Formulations

For the most part, formulas for total properties will be presented. It is straight-forward to derive the formulas for monochromatic properties, since monochromatic properties are simply total properties which apply to a single wavelength band. *Figure 2.2* will be used in our derivations.

The energy flux from the surface into the unit hemisphere above the surface can be found by integrating equation (2.3) over the entire hemisphere:

$$q = \int_{hemisphere} I \cos\theta d\omega \tag{2.5}$$

where

$$\int_{hemisphere} = \int_0^{2\pi} \int_0^{\pi/2} \tag{2.6}$$

Using equation (2.2) for a differential solid angle, the above integration becomes:

$$q = \int_0^{2\pi} \int_0^{\pi/2} I \cos\theta (\sin\theta d\theta d\phi) \tag{2.7}$$

If energy is <u>independent of direction</u>, then I is termed **uniform**, and the radiation is termed

diffuse. In this case the above equation reduces to:

$$q = I\pi \tag{2.8}$$

If I is uniform, from equations (2.1) and (2.3), the energy flux varies as $cos\theta$, and inversely as r^2, which is just **Lambert's law** for diffuse radiation [Siegel81].

For the above formulation **all** of the energy leaving a surface in an outward direction was considered. If considering only the original emission leaving a surface, the flux is the hemispherical emissive power E and the associated radiance is labeled I_e. Therefore:

$$E = \int_0^{2\pi} \int_0^{\pi/2} I_e \cos\theta (\sin\theta d\theta d\phi) \tag{2.9}$$

If a surface is a *diffuse emitter* (i.e. I_e is uniform), then

$$E = \pi I_e \tag{2.10}$$

For radiosity, which considers the original energy plus the reflected energy from a surface, with the associated radiance being labeled (I_{e+r}), the definition for B is:

$$B = \int_0^{2\pi} \int_0^{\pi/2} I_{(e+r)} \cos\theta (\sin\theta d\theta d\phi) \tag{2.11}$$

If a surface is a diffuse emitter and a *diffuse reflector*, then

$$B = \pi I_{e+r} \tag{2.12}$$

Next, the incoming energy is considered. If (I_i) gives the rate of energy incident upon a surface, per unit area of intercepting surface, normal to the direction (θ, ϕ), per unit solid angle, then the same relations (2.2) (2.5) apply where I_i replaces I. Thus *irradiation* is given as:

$$H = \int_0^{2\pi} \int_0^{\pi/2} I_i \cos\theta (\sin\theta d\theta d\phi) \tag{2.13}$$

and *diffuse irradiation* is given as:

$$H = \pi I_i \tag{2.14}$$

2.3.2 Relations

- Using the the **Theory of energy conservation** [Siegel81] and the above definitions, the following interrelations hold for the monochromatic and total properties:

$$\rho_\lambda + \alpha_\lambda + \tau_\lambda = 1.0 \tag{2.15}$$

$$\rho + \alpha + \tau = 1.0 \tag{2.16}$$

  Most gases have high τ and low α and ρ, while for opaque objects: $\rho + \alpha = 1$.

- From previous definitions for E, B and H properties, the following relation holds:

$$B = E + \rho H \tag{2.17}$$

2.4 Black and Graybody Radiation

A *perfect black body* will absorb all incident radiation regardless of spectral distribution or directional character ($\alpha = 1$). That is, only original emission leaves a surface. According to the *Stefan-Boltzmann Law for Blackbody Emissive Power* ([Sparrow78]), the *blackbody emissive power* (E_b) depends on temperature only. To simulate a blackbody environment, imagine an enclosure at thermal equilibrium. E_b is independent of orientation, and is equal to the irradiation (H), (which is constant in all places since it is all absorbed). The radiance of blackbody radiation (I_b) is uniform, and from equation (2.10) the blackbody radiation is:

$$E_b = \pi I_b \tag{2.18}$$

When considering radiation for more realistic *non-black surfaces*, at a given temperature, the emissive power (E_λ) may differ in amount and spectral distribution from that for a black body at the same temperature. That is, not all energy may be absorbed, and the emissive power may depend on the wavelength of the incident radiation. As such these types of surfaces may exhibit non-diffuse behavior such that the emission or reflected energy may not be constant.

Black and non-black surfaces are related by surface *emissivity* and *absorptivity*. The *monochromatic emissivity* (ϵ_λ) of a surface is defined as the non-blackbody emission divided by the blackbody emission

$$\epsilon_\lambda = \frac{E_\lambda}{E_b} \tag{2.19}$$

The *monochromatic absorptivity* (α) of a surface is the integral of absorbed irradiation divided by the total incident energy.

An *ideal gray body* is a special type of non-black surface, where the emissivity is independent of wavelength, or in other words, the ratio of E_λ to E_b is same for all wavelengths of emitted energy at a given temperature. Wavelength independence applying to absorption as well, the following relations are obtained:

$$\epsilon = \epsilon_\lambda \tag{2.20}$$

$$\alpha = \alpha_\lambda \tag{2.21}$$

According to **Kirchoff's Law** [Sparrow78], at thermal equilibrium and in an isothermal enclosure, the absorbed and emitted energies are equal, and, absorptivity equals the emissivity:

$$\alpha = \epsilon \tag{2.22}$$

as long as a gray surface is assumed, or surfaces are idealized as being gray.

2.5 Direct Interchange Between Surfaces

In this section the **direct** interchange of energy between two surfaces will be discussed, using the concept of a *form-factor*. This concept will be used again when discussing the **total** interchange of energy between all surfaces of an environment.

To start, a definition of the environment is required. The concept of an enclosure, is thus introduced. An *enclosure* is defined by a set of surfaces enclosing a space, such that one or more surfaces may or may not be material surfaces (e.g. an open window), and each of these surfaces has radiation properties and surface temperatures associated with them. Assuming an enclosure, a *form-factor* (F) defines the fraction of energy leaving a given surface that arrives at a second surface directly, and is found by determining the exchange between two differential areas on each surface and then integrating over both surfaces. If *Lambertian surfaces* are assumed, then a form-factor is dependent **solely on geometry**.

Referring to *Figure 2.3*, let us assume there are two surfaces i and j, with respective areas A_i and A_j, and respective differential areas on each surface dA_i and dA_j . From equations (2.1) and (2.3),

$$d\omega = \frac{cos\theta_j \, dA_j}{r^2} \tag{2.23}$$

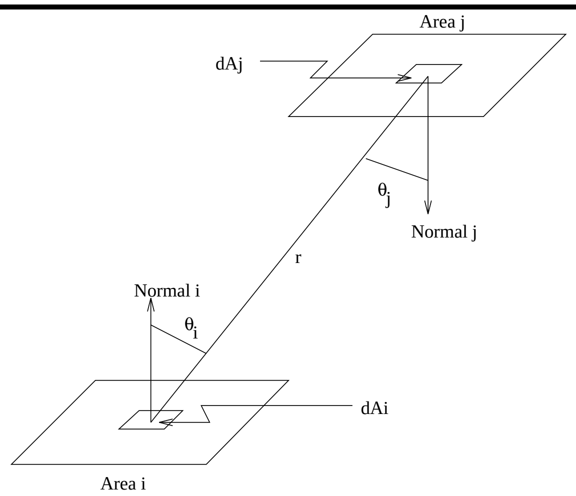

Figure 2.3: Form-factor geometry

$$dq = I_i cos\theta_i d\omega \tag{2.24}$$

the fraction of flux subtended by dA_j, as seen from dA_i is

$$dq = \frac{I_i cos\theta_i \cos\theta_j dA_j}{r^2} \tag{2.25}$$

Since dq is the portion of flux from dA_i intercepted by dA_j, it can be rewritten as:

$$dq = dq_{i-j}/dA_i \tag{2.26}$$

where q_{i-j} is the energy leaving all of A_i that strikes all of A_j. Therefore q_{i-j} is found by integrating over both areas of i and j

$$q_{i-j} = \int_{A_i}\int_{A_j} I_i \left(\frac{cos\theta_i \cos\theta_j}{r^2}\right) dA_j dA_i \tag{2.27}$$

If the radiosity of the originating surface (B_i) is uniform over that surface, the total energy leaving A_i which strikes A_j is

$$F_{i-j} = \frac{q_{i-j}}{A_i B_i} \tag{2.28}$$

If the radiant energy is diffuse then I_i is uniform, and from equation (2.12) which states: $B_i = \pi I_i$,

$$F_{i-j} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \left(\frac{cos\theta_i \cos\theta_j}{\pi r^2} \right) dA_i dA_j \tag{2.29}$$

F_{i-j} is called the *diffuse form-factor or configuration factor*. Note that equation (2.29) applies only for diffuse surfaces, while equation (2.28) is generally applicable. Also, F_{i-j} in general is not equal to F_{j-i} which is given as:

$$F_{j-i} = \frac{1}{A_j} \int_{A_j} \int_{A_i} \left(\frac{cos\theta_j \cos\theta_i}{\pi r^2} \right) dA_i dA_j \tag{2.30}$$

using a similar derivation as the one given for F_{i-j}.

2.5.1 Properties of Form-Factors

Assume that the interior of an enclosed space is subdivided into n finite surfaces with areas A_1, A_2 to A_n. In this section, a surface i will be called the **receiving surface** and a surface j the **transmitting surface**. Let surface i with area A_i be divisible into subareas $A_{i_1}, A_{i_2}, \ldots$ to A_{i_n} where $\sum A_{i_n} = A_i$. Each of these subareas may be of differential size, denoted dA.

1. *Reciprocity Property* : From equations (2.29) and (2.30), the following equalities may be derived:

$$A_i F_{i-j} = A_j F_{j-i} \tag{2.31}$$

$$A_i F_{i-dj} = dA_j F_{dj-i} \tag{2.32}$$

   and

$$dA_i F_{di-dj} = dA_j F_{dj-di} \tag{2.33}$$

   This is provided the flux distribution is diffuse, and for finite surfaces, the magnitude of the flux does not vary along the surface.

2. *Additive Property* : From equations (2.29) and (2.28) the following relation is obtained:

$$A_i F_{i-j} = \sum_n A_{i_n} F_{i_n-j} \tag{2.34}$$

   From reciprocity the following is derived:

$$A_j F_{j-i} = \sum_n A_j F_{j-i_n} \tag{2.35}$$

which implies that

$$F_{j-i} = \sum_n F_{j-i_n} \tag{2.36}$$

If the transmitting surface is subdivided, the form-factor for that surface with respect to the receiving surface is not the sum of the individual form-factors, as is the case when the receiver is subdivided. If both surfaces are subdivided into n and m parts respectively, then

$$A_i F_{i-j} = \sum_n \sum_m A_{i_n} F_{i_n - j_m} \tag{2.37}$$

3. *Enclosure property*: This property states that from our assumptions, n equations of the following form may be written, one for each surface:

$$\sum_{j=1}^{n} F_{i-j} = 1 \tag{2.38}$$

for $i = 1, 2..., n$. The above equation implies that no energy is lost in the enclosure. Note that some of the surfaces may 'see' themselves if they are concave (i.e. $F_{i-i} \neq 0$ for some surface i), and conversely if the surface is convex $F_{i-i} = 0$.

2.5.2 Calculating the Form-Factor

The calculation of form-factors for a pair of surfaces labeled i and j involves the following steps:

- The setup of a coordinate system of reference for each surface, and the location of dA_i and dA_j on the surfaces in terms of these coordinates.
- Calculating the angles and distance between the differential areas.
- The areas themselves are represented in terms of differentials of their coordinate system, which implies a two fold integral for each of the double integrals, implying a four-fold integral to solve.

For even the simplest geometries, the evaluation of the 4-fold integral is quite complicated. Currently, there are sets of form-factors already calculated for certain specific geometries of surfaces (e.g. [Siegel81]). By using some of the properties mentioned above, it is sometimes possible to reduce to these simpler cases to solve more complicated geometries. As well, mathematical methods such as contour integration has been used to solve the integrals (e.g. [Sarofim67]).

2.6 Radiation Exchange Among Gray Body Surfaces

The first type of environment to consider is one containing surfaces with graybody characteristics, which will be called surfaces with *diffuse characteristics*.

Assuming that a set of surfaces forms a complete enclosure, the following model is presented:

1. Each surface is isothermal, or are surfaces subdivided into sub-surfaces that are small enough to be isothermal. Also each surface is a diffuse emitter, and reflector.
2. Emitted and reflected radiation are directionally indistinguishable, therefore there is no need to treat them separately.
3. For a thermally opaque surface i, the radiosity is equal to the original emission from the surface plus any reflected incident radiation.

$$B_i = E_i + \rho_i H_i \tag{2.39}$$

   Radiosity is constant along any surface, and thus form-factors are independent of the magnitude, and surface distribution of the radiant energy flux.

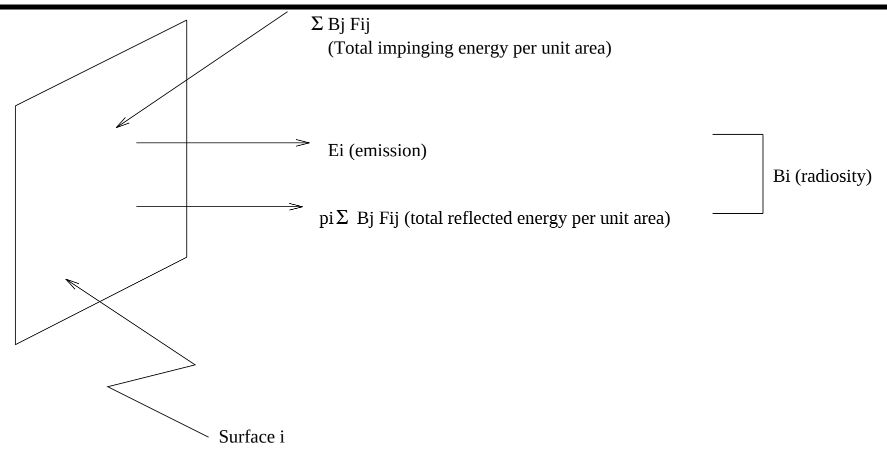

Figure 2.4: Radiosity for Gray Body Surfaces

In this model, the incident radiant flux H_i is given as

$$H_i = \sum_{j=1}^{N} B_j F_{i-j} \tag{2.40}$$

and therefore the radiosity of surface i is

$$B_i = E_i + \rho_i \sum_{j=1}^{N} B_j F_{i-j} \tag{2.41}$$

(See *Figure 2.4*).

Rarely, if ever will there be uniform radiosity on each surface, but this assumption is not required if differential surface areas are used to perform the computation. This allows for a continuous variation of radiosity, of surface temperature (hence emittance) and thus surface heat flux. The radiosity equation is redefined as:

$$B_i(x_i, y_i) = E_i(x_i, y_i) + \rho_i(x_i, y_i)H_i(x_i, y_i) \tag{2.42}$$

such that surface position (x_i, y_i) is now included. Using the corresponding replacement for H_i:

$$H_i(x_i, y_i) = \sum_{j=1}^{n} \int_{A_j} B_j(x_j, y_j)F_{di-dj} \tag{2.43}$$

the radiosity at any point (x_i, y_i) on surface i is

$$B_i(x_i, y_i) = E_i(x_i, y_i) + \rho_i(x_i, y_i) \left(\sum_{j=1}^{n} \int_{A_j} B_j(x_j, y_j)F_{di-dj} \right) \tag{2.44}$$

The total radiosity may be found by integrating over all points (x_i, y_i) on the surface i.

Equations like (2.41) and (2.44) are written for each of the n surfaces of the enclosure, to obtain a set of linear inhomogeneous equations of a general form given as:

$$\begin{bmatrix} 1-\rho_1 F_{1-1} & -\rho_1 F_{1-2} & \dots & -\rho_1 F_{1-n} \\ -\rho_2 F_{2-1} & 1-\rho_2 F_{2-2} & \dots & -\rho_2 F_{2-n} \\ . & . & \dots & . \\ . & . & \dots & . \\ -\rho_n F_{n-1} & -\rho_n F_{n-2} & \dots & 1-\rho_n F_{n-n} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ . \\ . \\ B_N \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ . \\ . \\ E_N \end{bmatrix} \tag{2.45}$$

for the n unknown radiosities B_1 to B_n, which was initially solved using matrix methods such as Gaussian Elimination ([Sarofim67]).

2.6.1 Wavelength Dependent Properties

The gray body model given is close to being realistic if confined to a certain wavelength level, but will give errors if it encompasses different wavelength spectrums. One way to make an exact account of wavelength dependence of radiation properties is by performing interchange calculations monochromatically, then integrating over the entire range of wavelengths. The problem is that there is not enough spectral detail known for most materials and the computation time may be lengthy. The commonly preferred choice is to break down the spectrum into finite bands of wavelength range, and then perform the calculations.

Chapter 3

Extensions and Approximations

In this chapter extensions to the gray body model will be examined to include specular surfaces and participating media, and some approximation methods for computing form-factors are given.

3.1 Radiation Exchange for Specular and Diffuse Environments

The first extension is the incorporation of surfaces with directional or *specular* reflective characteristics, using the *exchange factor equation* ([Sparrow78]).

Basically, specular and diffuse components of a surface are included by using an approximation of real hemispherical reflection by splitting reflection up into specular (ρ^s) and diffuse (ρ^d) components:

$$\rho = \rho^s + \rho^d \tag{3.1}$$

The radiosity equation, as before, is given as

$$B_i = E_i + \rho_i H_i \tag{3.2}$$

but H_i is now split into components from diffuse and specular surfaces as follows:

$$H_i = H_{i,diffuse} + H_{i,specular} \tag{3.3}$$

Letting N_d be the number of diffuse surfaces out of n total surfaces:

$$H_{i,diffuse} = \sum_{j=1}^{N_d} B_j e_{i-j} \tag{3.4}$$

and

$$H_{i,specular} = \sum_{j=N_d+1}^{n} B_j e_{i-j} \tag{3.5}$$

The *exchange factor* e_{i-j} is given as

$$e_{i-j} = f_0 + \rho^s_{1,1} f_1 + \rho^s_{1,2}\rho^s_{2,2} f_2 + \rho^s_{1,n} \cdots \rho^s_{n,n} f_n \tag{3.6}$$

where f_0 represents the direct transport of energy from surface i to surface j. In this model, e_{i-j} in equation (3.4) is just f_0 where $f_0 = F_{i-j}$. The rest of the terms represent the transport due to specular interreflections. In general a term like $\rho^s_{1,n}\rho^s_{2,n}\rho^s_{3,n} \cdots \rho^s_{n,n} f_n$ represents the fraction of the radiant energy leaving surface i and arriving at surface j after going through n intervening specular reflections. $\rho_{k,n}$ gives the specular reflectance of the surface in which the kth of the n specular reflections occurs, and the product of $\rho_{k,n}$ takes into account all specular reflections.

Exchange factors have similar rules to form-factors:

1. $A_i e_{i-j} = A_j e_{j-i}$
2. $A_i de_{i-dj} = dA_j e_{dj-i}$
3. $dA_i de_{di-dj} = dA_j de_{dj-di}$

The above rules apply only when the surfaces involved are gray.

3.2 Radiation Exchange in the Presence of Participating Media

The last extension to the model will be the inclusion of participating media. Our discussion will be restricted to gases. Gases may cause absorption, emission, or scattering of energy traveling though them, resulting in the energy being attenuated, enhanced or redirected before exiting the gas. Some new constructs are required to allow this change in energy to be taken into account.

First, some basic terminology is given:

- *isotropic* means independent of direction

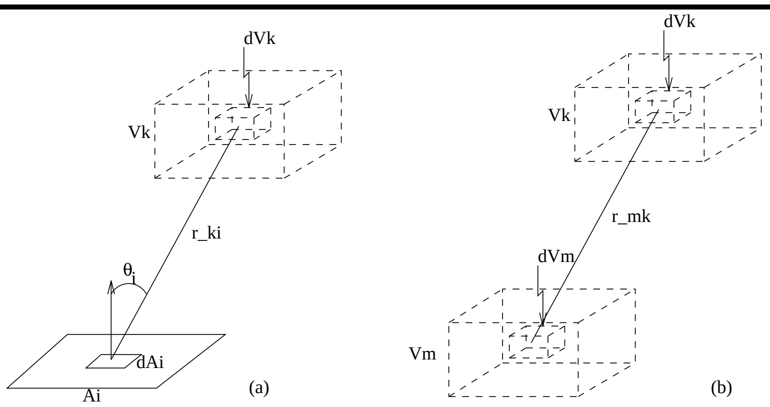

Figure 3.1: a) Surface to volume geometric factor b) Volume to volume geometric factor

- *Volumetric absorption* (κ_a) describes how much incident radiation is lost by traveling through a volume of media, and usually depends on the density of the media, the wavelength of the incident radiation, and the thermodynamic state of media.
- *Volumetric scattering* (κ_s) is defined as any change in the direction of incident energy traveling through a media. *Coherent scattering* is assumed, wherein no change in the wavelength of the energy results due to scattering.
- *Extinction Coefficient* (κ_t) is given as the sum ($\kappa_s + \kappa_a$).
- The *scattering albedo* (Ω), is the volume analog to the reflectance (ρ) for surfaces and is the ratio of κ_s to κ_t.
- *Volumetric transmittance* ($\mathcal{T}(s)$) is the fraction of energy that is neither absorbed nor scattered in traveling some distance s, through a volume of media.
- *Volumetric emission* (J) defines the original emission within the media.

There are various methods for solving this problem [Sarofim67] [Siegel81]. The **zonal method** [Sarofim67], will be presented, which assumes that gases are discretized into volumes or *zones* of gas. Each volume has isotropic volumetric emission, absorption, and scattering properties, and non-gas surfaces have diffuse surface emission and/or reflection. Volume to volume, and volume to / from surface radiosity exchange in this method will be outlined.

The equation for the radiosity at surface i with area A_i, emission E_i and reflectance ρ_i is given as:

$$B_i A_i = E_i A_i + \rho_i \left(\sum_j^s B_j \underline{S_i S_j} + \sum_k^v B_k \underline{V_k S_i} \right) \tag{3.7}$$

where s and v are the number of surfaces and volumes respectively, $\underline{S_i S_j}$ is the geometric factor describing the exchange of energy between surfaces i and j, similar to the previously described form-factors, and $\underline{V_k S_i}$ is a geometric factor describing the exchange of energy between volume k and surface i. These factors are given as:

$$\underline{S_i S_j} = \int_{A_i} \int_{A_j} \frac{\mathcal{T}(r_{ij}) \cos\theta_i \cos\theta_j \, dA_i dA_j}{\pi r_{ij}^2} \tag{3.8}$$

$$\underline{V_k S_i} = \int_{V_k} \int_{A_i} \frac{\mathcal{T}(r_{ki}) \kappa_{t_k} dV_k \cos\theta_i \, dA_i}{\pi r_{ki}^2} \tag{3.9}$$

For a volume (k), with volume V_k, radiosity B_k, emittance E_k, and albedo Ω_k, the defining equation for radiosity is:

$$4\kappa_t B_k V_k = 4\kappa_a E_k V_k + \Omega_k \left(\sum_{j}^{s} B_j \underline{V_k S_j} + \sum_{m}^{v} B_m \underline{V_m V_k} \right) \tag{3.10}$$

where $\underline{V_m V_k}$ is the geometric factor for two volumes, and is given as

$$\underline{V_m V_k} = \int_{V_k} \int_{V_m} \frac{\mathcal{T}(r_{mk}) \kappa_{t_m} \kappa_{t_k} dV_k dV_m}{\pi r_{mk}^2} \tag{3.11}$$

The geometries for $\underline{V_k S_i}$ and $\underline{V_m V_k}$ are shown in *Figure 3.1.*

3.3 The Unit-Sphere Method

This method, developed by Nusselt [Nusselt78], gives an geometric equivalent to a form-factor. The computation is as follows:

1. Place a hemisphere of radius r oriented around the normal of a differential area dA_i, and project a second differential area dA_j, located distance s from dA_i, onto the hemisphere to get a projected area:
$$dA'_j = \frac{dA_j cos\theta_j r^2}{s^2} \tag{3.12}$$
2. Orthographically project dA'_j onto the base of the hemisphere to get an area:
$$dA''_j = dA'_j cos\theta_i = \frac{dA_j cos\theta_i cos\theta_j r^2}{s^2} \tag{3.13}$$

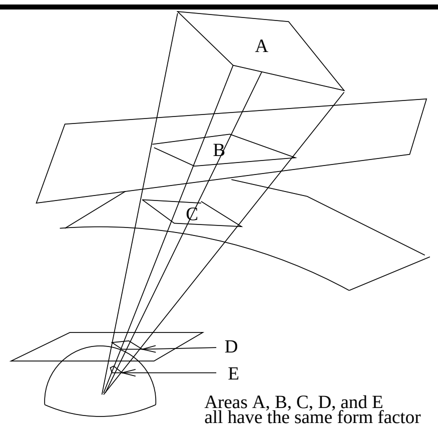

Figure 3.2: Nusselts Analog

3. The ratio of dA''_j to the base of the hemisphere is given as:

$$\frac{dA''_j}{\pi r^2} = \left(\frac{dA_j cos\theta_i cos\theta_j r^2}{s^2}\right)\left(\frac{1}{\pi r^2}\right) = \frac{dA_j cos\theta_i cos\theta_j}{\pi s^2} = F_{d1-d2} \tag{3.14}$$

which is the form-factor for two differential surfaces.

If the radius of the hemisphere (r) is 1, then the projected area dA'_j is the definition of a differential solid angle ($d\omega_i$) for surface i. Furthermore, all patches dA_j have the same form-factor if, when projected, they subtend the same solid angle $d\omega_i$, when viewed from dA_i and are positioned at an angle θ_i with respect to the normal of dA_i. This fact holds true for projections onto any surrounding surface. Therefore, any surrounding surface may be used in place of a hemisphere to derive the form-factor. This fact will be referred to as *Nusselt's analog.* (See *Figure 3.2*)

3.4 Monte Carlo Approach

Monte Carlo methods are statistical numerical methods [Sparrow78], that can be applied as a sampling technique to reduce the number of samples required to compute a form-factor.

In this approach, energy exchange is modeled by following the progress of discrete amounts ("bundles") of energy as they interact with various surfaces. The physical problem is modeled as a probability function using random numbers to "play". Form-factors are computed as the fraction of the total number of energy bundles emitted from a surface that are incident upon a second surface. The bundle paths and their histories are computed by a Monte Carlo method.

To demonstrate the ideas behind the approach a simple example from [Sparrow78] is given:

> A surface i is selected to emit energy and a number N is selected for the number of bundles to emit. Probability functions are used to determine the probability that a bundle will be emitted from this surface, and of emitting in a given direction. Each emitted bundle, has it's path traced. If a bundle strikes a second surface j the amount of energy that is absorbed or reflected is determined by picking random numbers from a uniform distribution. Each bundle path is followed until the bundle is completely absorbed. This process is followed for all other surfaces.

Note that this approach will require a large number of random numbers in order to give statistically satisfiable results, and even so it cannot guarantee convergence to the correct result.

3.5 The Image Method

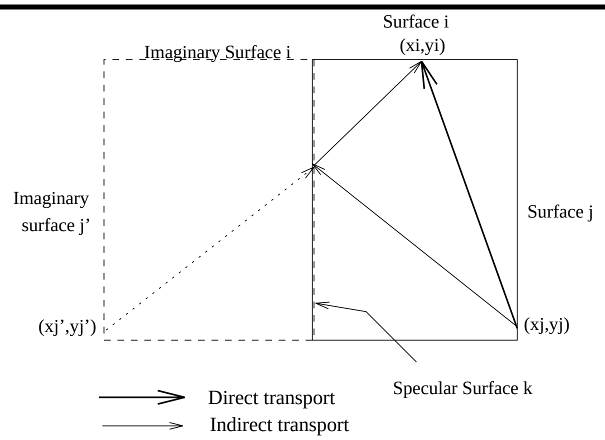

Figure 3.3: Image Method

The task of computing an *exchange factor* involves keeping track of the attenuation of original emission from a surface i as it is reflected off a number of other surfaces, until it strikes a second surface j. In this section, a method called the *image method* is presented for computing this factor assuming that the reflectance of all surfaces is ideal. Ideal reflectance is when any incoming directional irradiation is reflected out in exactly one outgoing direction for each incoming direction.

Let us consider an environment like the one shown in *Figure 3.3*. There are two surfaces i, and

j, that we wish to compute the energy exchange for. Direct transfer of energy can be accounted for using the form-factor F_{i-j}. However, some energy from j may reflect off a third surface k before reaching i. To take this indirect energy into account, an imaginary world can be formed (denoted by the dashed lines) which is the reflection of the enclosure in surface k. To an observer at point (x_i, y_i) on surface i, the energy appears to originate from point $(x_j, y_j)'$ on the imaginary surface j'. If the specular reflectance of surface k is ρ_k^s, then the effective energy from imaginary surface j' is given as $\rho_k^s F_{i-j'}$. Therefore the exchange factor equation from surface j to i may be given as:

$$e_{i-j} = F_{i-j} + \rho_k^s F_{i-j'} \tag{3.15}$$

For multiple specular reflections, this idea may be extended. If the path of energy from some original surface j is followed, for any intermediate specular surfaces the energy strikes, an image world can be formed. The energy can be attenuated by the reflectance of each specular surface (ρ), by successively updating the form-factor by multiplying by each reflectance.

Chapter 4

Survey

4.1 Introduction

In recent years, there has been a growing interest, in the field of computer graphics, into using various techniques to solve the global illumination problem using the radiosity method. The focus of this chapter will be to provide a comparative examination of some of the more important aspects of these techniques. Our discussion will be ordered along the lines of the steps required to produce a model with physically based illumination starting with an initial environment description. A basic outline of these steps is as follows:

1. Generate a model of the environment using applicable constraints. This includes identifying the basic geometric unit, which, will loosely be called an "object" in our initial discussion.
2. Decide on the approaches for determining visibility between objects, and computing form-factors.
3. Decide how to capture sufficient radiosity sampling density and distributions within the restrictions imposed by the geometric model, and form-factor choices.
4. Choose the mechanism for propagating energy. These mechanisms will be referred to as **radiosity solutions**.
5. Determine the scheme for rendering the model.
6. Take into account the possibility that objects in the environment may move, or that the position that the model is viewed from may change.
7. Determine how to compute solutions in a parallel manner.

This breakdown is only assumed so as to be roughly analogous to how radiosity values are computed by heat transfer methods. Naturally, these points need not be performed sequentially, as all choices may effect all other choices. To reduce overlap in our discussion, for the most part mutual dependencies will only be presented in detail the first time they are discussed.

For each of the above steps, the following applicable points will be covered:

- Assumptions: Assumptions made concerning given aspects in the area.
- Cost: Computation and storage cost.
- Physical basis: Physical effects that can be simulated.
- Coherence: Use of appropriate coherence properties. *Coherence* is the extent to which parts of an environment exhibit local similarities.
- Compatibility: Compatibility between algorithms from different areas.
- Accuracy: How accurate are the solutions compared with physical simulations.

Appendix B contains brief explanations for some general terms that will be referred to in this chapter. For more extensive descriptions, we refer you to the references given.

4.1.1 Assumptions and Generalizations

In the field of heat transfer, a larger class of problems is considered than what is considered for current computer graphics implementation of the radiosity method. Some simplifying assumptions are now introduced to focus our attention on the actual problem to solve.

In general the environment may not be an enclosure and hemispherical properties may not always be upheld (i.e. the law of energy conservation may not hold with the environment). All surface and radiative properties are independent of temperature, and such properties may be specified and computed using a discrete set of wavelength bands. Within each band, any wavelength dependent properties are constant, and there is no interdependence between wavelength bands. The bands will generally represent a colour space such as RGB. For rendering, computed radiosity values are used as colour values.

Geometrically, surfaces have no thickness, and may be discretized into polygons of some fixed shape and size, with uniform surface properties.

4.2 Modeling

Before performing any radiosity algorithm on a given scene, the scene must be modeled. This section will cover some techniques used to accomplish this modeling task. Included are techniques to polygonalize a scene to produce boundary representations, including the use of space subdivision and half-space subdivision. In addition, volume representations, and some non-polygonal surface representations will be presented. The focus of the discussion will be on constraints that must be enforced in a model due to the representations used, restrictions in the choice of representation used due to the nature of the radiosity algorithm, and restrictions on the radiosity algorithm to use due to the choice of representation. Our discussion will also examine some extensions that have been introduced to handle shortcomings with using data structures that are not specifically geared towards the radiosity problem.

The first constraints to examine are *modeling constraints*. That is, some important information must be captured in order for our models to be complete. This not only means being able to represent the scene in a geometrically consistent manner ([Mantyla88]), but also containing sufficient information to satisfy various algorithm requirements, such as for form-factor and radiosity computation, and rendering.

Referring to *Figure 4.1*, the following are desirable modeling constraints for radiosity algorithms:

- **No T-vertices**: If two adjacent polygons fail to share a vertex that lies along their common edge, such a vertex is called a T-vertex. As shading discontinuities may result, they should be removed.
- **No light or shadow leakage** : These are artifacts, due to interpolation of incorrect values through a boundary, and should be removed.
- **Normal consistency**: If counting vertices of any surface in a given direction, the normal should always point in a fixed direction with respect to the counting order.
- **Well-shaped surfaces**: The aspect ratio of the radius of the inscribed circle of a surface to the radius of the circumscribed circle of a surface, is close to one ([Frey87]). **Irregular-shaped surfaces** are not well-shaped.

Secondly, in terms of *geometry*, models should contain sufficient information, so that there are no ambiguous points, and each surface has a unique front and back, with a unique outward normal per surface. If normals are not given, normals should be derived in a consistent manner (normal consistency). Cases of coplanar faces, and interpenetrating volumes require methods to handle them.

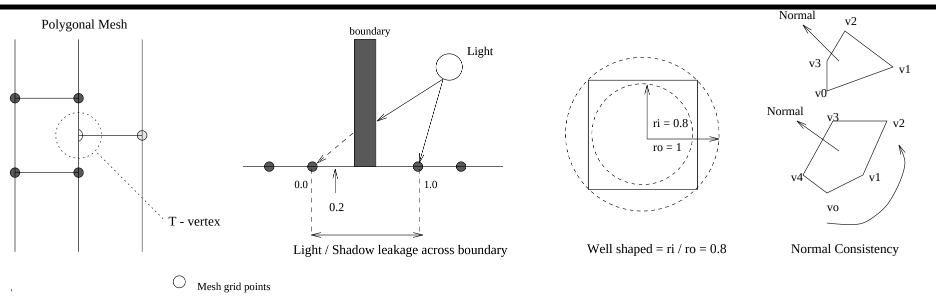

Figure 4.1: Mesh definitions: (from left to right) a) T-vertex: Circled vertex is not shared by polygon on the left. b) Light leak: In 2-dimensions, the interpolated values along the line, between the grid point with value 0 and the boundary will be greater than one, which is incorrect. Values between the boundary and the point with value 1 will have values that are too small due to interpolation. c) Well-shaped: The best ratio is 1 to 1 for the inscribed and circumscribed imaginary circles, respectively, for a given shape. The ratio is shown for an example polygon. d) Normal consistency: For example polygons shown, counter-clockwise vertex counting gives the same normal direction for all polygons.

Third, in terms of *radiosity sampling concerns*, to store radiosity values, a set or mesh of sample points is usually used. Concerns include: a) discontinuities across surface boundaries, b) maintaining well shaped faces for sufficient sample density, c) appropriate sample density, resulting from sample point distribution, d) explicit representations of intersecting objects to avoid light or shadow leakage, and e) for each surface, whether only the exterior of the surface radiates energy, or whether both sides are allowed to radiate energy must be determined.

Finally, in terms of *display requirements*, it is desirable to have no T-vertices via either meshing on the same surface or between surfaces, and well-shaped faces to avoid display artifacts if using Gouraud shading ([Hall89]).

4.2.1 Boundary Representations

Introduction

The first representation for modeling scenes, that will be examined, are *boundary representations* or *breps*. This approach represents a model in terms of surface boundaries which are composed of vertices, edges and faces. Often, boundary representations are used to produce approximations of real surfaces, by discretizing surfaces into explicit representations in the form of polygonal meshes.

Before discussing any of the approaches to creating boundary representations, let us examine some basic problems with these representations. The first major problem, when dealing with discretized surfaces, is obtaining a polygonal breakdown such that polygons have uniform radiosity. Several guesses have been proposed for manually performing this task ([Goral84], [Cohen85]). Unfortunately, an accurate first guess cannot be given since the breakdown desired is the geometry that would ideally be derived after computing the solution. Generally, either a very fine initial mesh is given or a rough initial guess is given along with some adaptive mesh subdivision technique for refining the mesh.

Other problems with this representation include the following:

1. Meshes may only be geometric approximations of real surface geometries.
2. Meshing is either manually done, or has problems when automated.
3. Restriction to convex or planar polygons, due to radiosity solution constraints.
4. Rendering artifacts may result if using interpolated shading.
5. Capturing rapidly changing gradients may require very fine meshing.
6. Remeshing on changes in surface or geometric properties.

These problems will be examined in the next section.

Approaches

Cohen presented the first subdivision hierarchy to capture surface subdivision ([Cohen86]). In this approach, objects are geometrically broken down into *surfaces*, *patches*, and *elements*. Surfaces are polygonal meshes chosen as the breakdown of an object. These meshes are composed of elements and patches. Elements are planar polygons, with uniform radiative properties, and may or may not share a common normal. Patches are convex polygons composed of one or more elements, and as such may have varying radiative properties. An example is shown in *Figure 4.2*. This method of *substructuring* has the following characteristics associated with using a hierarchy:

1. Being able to work to a desired level of detail.
2. Sharing of information between levels.
3. Capturing radiosity gradients via patch and element subdivision.
4. Saving on storage space, by subdividing surfaces only where needed.

In order to maintain well-shaped surfaces, a *quadtree* ([Foley90]) may be used to implicitly represent this hierarchy. For efficient (constant time) adjacency searching with respect to edges,

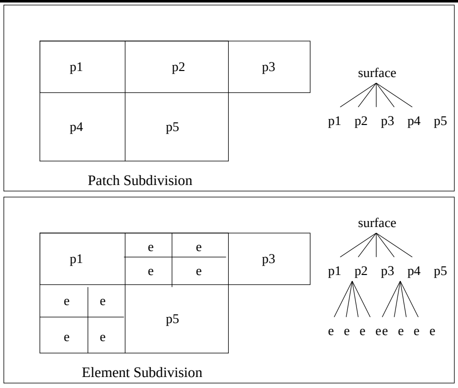

Figure 4.2: Substructuring: This diagram shows the structure hierarchy given in [Cohen86] consisting of elements, patches, and surfaces.

a *winged-edge data structure* ([Foley90]) is typically used. A possible disadvantage is restricted usage of other models, due to the inability to represent certain objects that can be represented using other models (e.g. *constructive solid geometry* models ([Mantyla88])).

[Baum91] also uses a winged-edged structure for adjacency, but a *tri-quad tree* (triangle-quadrilateral tree) is used to represent the subdivision hierarchy. Baum presents automatic mesh creation techniques which incorporates various modeling constraints. Problems addressed included: a) removing initial T-vertex discontinuities by joining surfaces that share edges (*ziplocking*); b) computing explicit object intersections to avoid light or shadow "leaks"; and c) solving the coplanar surface problem by dropping the surface with the smaller surface area. If surfaces are not opaque, there are potential problems with removing a surface that will effect energy transfer.

In order to create the initial patch mesh, a quadrilateral grid with fixed subdivision length is mapped to each surface, so that polygons will be well shaped without T-vertices. Meshes are aligned with edges of a surface. There are problems with potential overmeshing, and mesh alignment at surface boundaries (e.g. where a sphere intersects a cube). Since only geometric information is used to compute a uniform mesh, the appropriate mesh density may not be computed where needed. The initial element mesh is created by computing pixel-level visibility between all surfaces using a depth-buffer. This may produce considerable overmeshing, and inaccurate splitting boundaries, though is computationally faster than its analytic counterpart ([Campbell90]).

For further subdivision of the original mesh, bilinear subdivision is used to subdivide quadarilaterals and triangles, producing a tri-quad tree per surface. To maintain gradual mesh density changes subdivision is performed so that there is at most one level difference between nodes, producing a *balanced tree*. Removal of T-vertices is achieved by *anchoring*, wherein neighbours are split by connecting corners of neighbouring polygons to the T-vertex. Splitting is thus localized to prevent excessive meshing. Both of these constraints have not been maintained by other subdivision methods as seen in [Cohen86], [Campbell90], and [Hanrahan91].

[Haines92a] examines more realistic scenes, not just the axis-aligned box scenes usually given as test cases in most research. Similar meshing solutions are presented for mesh point placement. Various subdivision methods such as evenly spaced, constant distance, as well as research into "ether meshing" is attempted. The first is hard to compute for complex surfaces causing potential misalignment of mesh points at edges between polygons, the second may produce misaligned meshes if taken from surface end points, and the third method produces arbitrary non-uniform meshing.

4.2.2 Voxel Representations

For modeling volumes of space, a space subdivision scheme may be used. *Space subdivision* involves subdividing the space of an environment either uniformly or non-uniformly, with various criteria as to how granular the subdivision should be. A subdivided volume of space is defined as a *voxel*. There is currently only one paper ([Rush87]) that uses this form of subdivision in order to model participating media. In order to represent non-uniform volumetric breakdowns of gases, an octree structure ([Samet90]) is used, wherein each non-empty voxel is equivalent to a volume of gas.

4.2.3 Space Subdivision

Space subdivision has also been used to exploit spatial coherence in environments.

Wang ([Wang90]) makes use of octrees to subdivide the scene until there is either one polygon per voxel, or the size of the voxel is less than some tolerance (t_1), or the size is less than some second tolerance (t_2) but more than one polygon is allowed per voxel. As opposed to [Rush87], the aim of using an octree is not to capture the geometry of the scene, but instead for more efficient form-factor computation. As a result, polygons may be split into irregular shapes. What are good subdivision tolerance values are never given, nor reasons why this subdivision scheme gives good shadow subdivision or sharp object corners. Wang only assumes to interactively split surfaces along shadow boundaries.

Airey ([Airey90]) notes that the complexity of an environment is independent of the complexity of shading and the amount of detail for the model. Suggested is the use of *model space subdivision* to form *cells* such that each cell has a possible set of polygons visible to all viewpoints in the cell. Binary subdivision planes are used to partition, based on the priority: (1/2 occl + 1/3 balance + 1/5 split), where *occl* is how much occlusion occurs, *balance* is how evenly the scene is subdivided, and *split* is how many objects or surfaces would be split in the process. The subdivision is only suitable for axis aligned boxes, and the subdivision criteria is based on a experimental heuristic. From experimental results this method requires about 20 percent more memory to store this visibility precomputation. Again, as in [Wang90], splitting may produce irregular shaped polygons.

4.2.4 Half-space Subdivision

When subdividing a surface, subdivision edges should be aligned as close as possible to the surface radiosity gradient. If bilinear subdivision is arbitrarily used for a surface or volume of space, T-vertices, excessive meshing and only approximate solutions may result. An example is given in *Figure 4.3*.

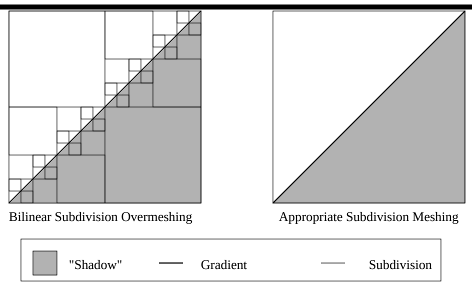

Figure 4.3: Subdivision incompatibility. Note excessive overmeshing if using bilinear subdivision on the left, while only one subdivision split is required in the scheme on the right.

[Campbell90] addresses these problems by using shadow volumes to split surfaces. *Binary space partitioning trees (BSP trees)* ([Foley90]) are used to compute visibility with respect to light sources, and to perform splitting. This allows for more accurate radiosity boundaries in the subdivision, and fixes light leaks at intersecting surfaces. BSP trees, in general, are computationally expensive especially if the geometry of the scene changes. This method also retains the problem of T-vertices, and may produce irregular shaped polygons due to plane cutting. Currently, there is no good subdivision stopping criteria, resulting in possible overmeshing.

4.2.5 Alternate Representations

To close this section, some outstanding issues are presented. First of all, there has been little research into modeling enclosures, or at least enclosures that are not rectangular boxes. Haines ([Haines92a]) presents the question as to how to effectively simulate the sun. A meshed hemisphere is suggested, similar to [Nishita86], which splits the hemisphere into longitudinal bands of constant radiosity. These approaches may require a large number of extra steps to compute results, since energy must be propagated from each mesh component.

Secondly, [Airey90] suggests to use either parametric representations or some other form of algebraic representation to represent surfaces as opposed to treating surfaces as independent polygonal entities. [Heckbert90] proposed the usage of the original geometry of objects to avoid polygonization altogether. In general using a higher order, less explicit representation may reduce the computational complexity, by reducing the number of entities to compute with. For example, if ray tracing is used, calculations such as ray-object intersection testing may be reduced. Ideas along this line are explored in greater detail in the section on form-factor acceleration techniques for ray tracing and in our implementation.

4.3 Form-Factors

The computation of form-factors is one of the most fundamental operations to perform in a radiosity algorithm. As such, a great deal of research has gone into this problem. The task usually involves a geometric computation to derive how much of one patch's energy will be transported to the other. If the environment has objects that occlude other objects (*complex environments*), then the *visibility* between applicable patches must also be determined. There is currently a lack of research into algorithms geared towards determining *surface-to-surface* visibility accurately and economically, as opposed to *point-to-point* visibility, which is used in the algorithms presented here.

There are two basic categories of form-factor computations: Those that just determine the amount of energy to propagate, and those that incorporate this computation as part of energy propagation. Emphasis will be given to sampling visibility between a source and receiver patch, and how physically accurate the form-factor computation is. The *source patch* is the patch that is considered to be radiating energy to all other patches, which are the *receiver* patches.

Sections 4.3.1 to 4.3.5 will discuss the first category of form-factors, while section 4.3.6 will cover the second category from the viewpoint of a ray tracing methodology.

4.3.1 Pure Analytic Approaches

In this section approaches that compute form-factors using analytic methods, by basically computing line or contour integrals, will be considered.

Goral ([Goral84]) performs contour integration to derive form-factors by using *Stoke's Theorem*. This basic method can be found in most heat transfer texts ([Sarofim67], [Siegel81], [Sparrow78]), and is only applicable to simple environments which do not contain occluded surfaces.

Line integration is used in [Nishita85]. The method computes umbra and penumbra volumes to determine visibility as a preprocess, and then performs point to point sampling to compute the form-factors. Important aspects of this method are:

- Separation of the visibility computation from the form-factor computation.
- Computation of form-factors at the vertices of patches using the normals at those positions, for rendering purposes.

4.3.2 Depth-Buffer Approaches

In this section, approaches that determine form-factors using *depth-buffer* techniques will be considered. Included is the discussion of sampling problems associated with using a depth-buffer, approaches to reduce the effects of these problems, use of coherence and other acceleration methods, and how these approaches are able to capture both diffuse and specular direct energy transport.

In the original *hemicube method* ([Cohen85]), a hemicube instead of a hemisphere is used to compute form-factors, according to Nusselt's analog ([Nusselt78]), as shown in *Figure 4.4*. The advantage is the ability to determinate visibility in complex environments by using depth-buffer techniques. Basically a unit size hemicube is positioned at the center of the front face of a patch, such that the base of the hemicube is perpendicular to the normal of the patch. *Delta form-factors* are precomputed and stored for each grid cell of the unit hemicube to reduce computation time, though requires storage proportional to h^2, where h is the the hemicube resolution. A delta form-factor is the form-factor computed from the center of the base of the hemicube to a cell on the grid of the hemicube. The technique involves determining what surfaces are visible through a frustrum formed from the center of the patch with each face of the hemicube. To determine which surface is visible at a given grid cell, item buffering ([Weghorst84]) is used, which records a patch's identifier at a grid cell if the patch is visible. The form-factor from a patch i (on which the hemicube resides) to any other visible patch j, is given as the sum of the delta form-factors for any cells with patch j's identifier recorded.

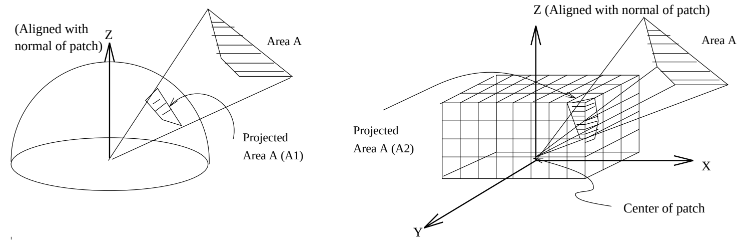

Figure 4.4: Hemicube Analog: Projected area A2 has the same form-factor as area A1 according to Nusselt's analog.

Using a hemicube inherits the problems found in standard depth-buffer visibility techniques, as well as producing new problems, mainly due to the use of discrete uniform sampling through the grid cells of the hemicube. These problems are as follows:

1. Since the grid resolution is non-adaptive either over or undersampling of the hemisphere may occur.

2. Uniform grid sampling may cause aliasing due to under or over estimations of surface projections. An example given by Haines ([Haines91b]) is the following: Suppose, the original projection of a polygon covers 6×6 cells. If the polygon is slightly offset, it covers 7×7 cells. There is thus a large change in projection size for a small change in position since *whole* grid cells are *counted*.

   This problem shows up as quite obvious artifacts for projections of small areas, in particular emitters, since this technique is insensitive to where the viewer is positioned. For example, imagine a door knob is lit with small emitter. The projection of the emitter may not cover all the cell centers, therefore black patches may be seen from viewpoint of the eye if the knob is viewed from a close distance ([Haines91b]).

3. The hemicube method performs its sampling from the *center of a patch*. This assumes that the visibility from the center applies for the whole patch, which may be quite erroneous. As noted in [Wallace89], form-factors may only be computed from finite patches, and thus cannot be computed using true surface normals at polygon vertices, where colour values are used for rendering.

4. No one has addressed the problem of the size and position of the hemicube with respect to the source patch. If the size or position of the hemicube is incorrect, surfaces that can be seen may be clipped, and surfaces may be included that can not be seen at all. This is especially true if concave patches are used.

What then, are its advantages ? Most proponents of this method ([Immel86] [Chen89] [Rush90b] [Baum90]) note that the method can use existing graphics hardware to perform computations such as back-face culling, projections, geometric transformations, clipping, and depth-buffering ([Foley90]).

The hemicube method has been explored quite extensively, resulting in many extensions to this technique. Some important extensions will be covered in the proceeding sections.

Form-Factor Approaches

The first extension was seen in [Cohen86], wherein form-factors are computed from elements to patches, as opposed to from patch to patch. Using the additive property for form-factors, patch form-factors are computed as area weighted averages of element form-factors. Cohen uses a *full-matrix (FM)* radiosity solution method. ([Goral84]).

The second extension was seen in the iterative *progressive refinement (PR)* radiosity solution method presented in [Chen89]. Chen computes the propagation of energy from one patch to all other patches and elements, instead of from all patches and elements to one patch. To maintain the computational cost of one hemicube per iteration, reciprocity and enclosure rules for form-factors are used.

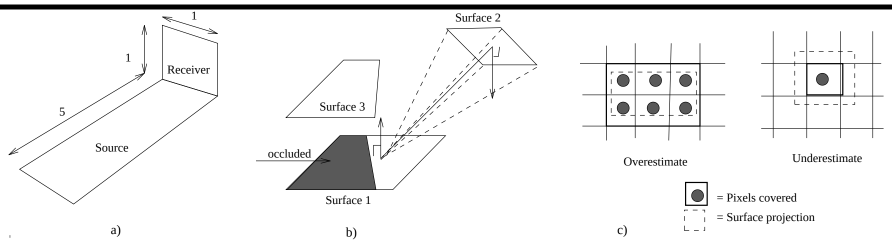

Figure 4.5: a) Large elongated area source to smaller receiver. b) Surface 3 is missed due to source undersampling at surface 1. c) Aliasing due to pixel counting of projections on uniform grid. (After [Baum89])

Referring to *Figure 4.5*, the following problems, due to assumptions made in the progressive refinement method, were presented in [Baum89]:

1. For valid form-factors it is required that the area of the source be very small with respect to the distance between the source and the receiver due to the use of the reciprocity property, and the hemicube approximation. This *proximity assumption* is broken much more often

using a PR method then when using a FM method, since form-factors are computed from source patches to (smaller) element receivers. Also noted was that the ratio of the area of the source to the receiver may be large, and sources may be elongated, giving less accurate form-factor values.

2. Since sampling is performed from one point on the center of the source, objects that occlude may be missed.

3. Area sources are reduced to point sources when computing patch to element form-factors.

4. FM and PR methods do not reach the same solution since one numerically integrates form-factors from receivers to sources, and the second assumes a constant form-factor from sources to receivers. This implies that no source subdivision is performed when shooting from sources, except when the shooting patch is large or very close to the receiver. That is, form-factor values become > 1, which is invalid.

5. Hemicube aliasing still exists.

[Baum89] provides some solutions by using an analytic contour integral form-factor approximation ([Sarofim67]) which considers the total area of the source and does not suffer if the distance separating the source and receiver is small, or the area of the source is large. There is still a problem in the fact that a hemicube is used to determine the visibility, and the projected areas of receivers, which are then used to compute the analytic form-factor from receivers visible to the source. Thus, any error associated with the hemicube projections is carried on to the form-factor calculation.

Baum ([Baum89] suggests subdivision of sources until they are either completely visible or non-visible from each receiver element in the environment, to avoid missing objects that occlude the source from the receiver. As with [Campbell90], the aim is to have visibility between the receiver and source accounted for, to avoid recomputing visibility upon subdivision.

From experiments, the solution is still only within 4 percent of a sample FM solution, as opposed to roughly 20 percent for the original PR method ([Chen89]).

Anti-aliasing

Given that a uniform grid is used to sample the hemicube, methods are required to remove or reduce artifacts due to aliasing. Suggested by Haines, Greenberg, Baum ([Haines91b], [SIG91], [Baum91]) and others, is to use (hardware assisted) anti-aliasing via an *accumulation buffer* or *a-buffer* [Haeb90]). This option has not been implemented. An alternative, suggested in [Airey90], is to use a *jittered hemicube* which is implemented in [Chen91b], by jittering the direction of the surface normal used for the hemicube. This idea is similar to using a jittered viewing frustrum ([Wallace87]), and has the same advantage of maintaining the speed associated with depth-buffering.

Coherence and Acceleration Techniques

Some techniques that have been presented to accelerate the computation of form-factors using depth-buffer techniques are now discussed.

The first methods, presented by Rushmeier, exploit *spatial coherence* and *pixel coherence* ([Rush90b]). Spatial coherence is exploited by using a user-defined fixed space subdivision to reduce the number of polygons sent to check on the hemicube. If possible, they suggest exploiting hardware by only sending shared vertices once per hemicube face, and to use hardware back-face culling. Only the first idea is tested. Pixel coherence is exploited by computing *cumulative form-factors* versus summing delta form-factors. Results of roughly 2.1 to 6.7 percent speedup are given, as it is faster to perform integer comparisons versus summations. If the number of surfaces is kept constant, the total suggested speedup is roughly proportionally to the hemicube resolution. From experimental scenes, the use of hardware roughly increases the speed for computing form-factors 40 times, as opposed to only using software.

Recker ([Recker90]) accelerated the process by using a single plane approximation of a hemicube. To sufficiently sample close to the normal of a patch, a two level uniform grid is used, with a finer grid centered around the normal. There is possible high perspective distortion (i.e. precision of depth computation is decreased) causing aliasing, and the problem of discontinuities at the boundary between the two levels of the grid. The claim is that there is a potential to save 80 percent in memory for the storage of item buffers, depending on how finely meshed the grid is, and the size of the grid. Increased computation and storage time results from keeping track of errors due to excluding form-factor computation through the sides, and larger areas to scan convert since the frustrum formed by the plane is greater than 90 degrees now.

Finally, [Beran91] uses a tetrahedral shape variation, to reduce computation cost as well, but with less perspective distortion than Recker's planar method. Basically, three planes are used to reduce the number of projection, hidden surface, and clipping calculations, but this requires slightly more storage for delta form-factors.

Generalizing Surface Properties

The original hemicube method ([Cohen85]) is only able to capture ideal diffuse direct transport. Some extensions to incorporate less restricted energy transport mechanisms will now be examined.

Immel ([Immel86]) was the first to incorporate specular effects in his *global cube* method, which uses a tesselated cube instead of a hemicube. Since the computation is done in world space as opposed to the coordinate system of surface, as with the hemicube, precomputation of delta form-factors is not possible. The worst aspect of this technique is that $O(c^2)$ storage of directional reflectance is required for every cube at every patch, c being the resolution of the grid for the cube, resulting in a $O(n^2c^2)$ matrix of form-factors to solve, n being the number of patches in the scene. In the worst case, the resolution of the cube may be to pixel levels for highly specular surfaces. Also, since sampling is performed at sparsely distanced points in the scene, sampling artifacts in images produced are quite obvious.

Shao ([Shao88]) uses a similar approach to Immel's but modifies the definition of a form-factor such that F_{i-j} stores incoming energy from all patches j to patch i in the hemicube. Unfortunately, form-factors become dependent on the emitter or energy distribution, specular patches still need to have very finely meshed hemicubes, and in the worst case $O(nh^2)$ extra storage is required, where h is the hemicube resolution, and n the number of specular patches.

Further extensions were seen in [Rush87], which uses a hemicube variation to compute the transport of energy between surfaces and volumes when participating media is present. The general formulation for these form-factors: $\underline{\text{SS}}$, $\underline{\text{SV}}$, and $\underline{\text{VV}}$ were given in *Chapter 2*. In [Rush86] and [Rush90a], hemicubes are used on the back side of a surface to simulate planar translucency effects, and the *image method* ([Sparrow78]) is used for ideal planar specular effects. Total effects are derived by linearly combining values from each computation. These methods are restricted to environments with planar surfaces, with only a small number of specular surfaces affecting diffuse surfaces. If the number of specular surfaces is large, the image method requires that the environment be "reflected" too many times.

Lastly, to incorporate "bumpy" surfaces, [ChenH90] used Blinn's idea ([Blinn78]) to perturb the normal of the surface for which the hemicube computation is performed with. Unfortunately, the formulation loses the information behind the perturbed hemicube, since it is assumed that the hemisphere seen by a perturbed and unperturbed hemicube is the same. As the degree of roughness is highly dependent on the meshing of the surfaces, this method only seems appropriate for large scale texturing, without undue time for computing hemicubes at every mesh point.

4.3.3 List-Priority Algorithms

As opposed to using a depth-buffer algorithm, others have used list priority algorithms ([Foley90]). For these algorithms, visibility is determined by considering objects in a front to back order, with respect to some viewing position, and "discarding" parts of objects that are behind what is currently visible.

Wang's technique ([Wang90]) computes an *octant priority table* as a preprocess, to determine depth priority with respect to a voxel. If the geometry and subdivision of a scene is constant this table need only be computed once. To determine visibility, a depth-sort algorithm ([Foley90]) is used, using the priority table to determine the sorting ordering of polygons. This approach, as with other single plane algorithms reduces the amount of comparisons to determine visibility. The shortcome of this approach is that the table is non-adaptive, due to the assumption that the subdivision of patches is already along appropriate boundaries.

[Campbell90] uses a two-step approach of first computing visibility, and then using Wallace's form-factor approximation ([Wallace89]). Initially, a volume is formed between the centroid of the source and the edges of the receiver closest to the source. Subsequent receivers are subdivided by this volume, which is successively refined as receivers are considered in a front to back order with respect to the centroid of the source. That is, if there is partial occlusion, then receivers are split along the planes defining the volume, and the planes of the volume are updated using the edges of non-occluded parts of the receivers. The major advantages of this approach is that no objects are missed and visibility boundaries are exact. The major drawbacks are computational complexity, restriction to objects polygonalized into planar polygons, potentially excessive splitting of polygons where such splitting is not required for capturing radiosity gradients, and other possible meshing problems, mentioned in the section on modeling.

4.3.4 Ray cast and ray tracing variations

In this section, using ray casting or ray tracing to perform sampling for form-factor computations will be considered. In general, ray casting / tracing has been used to provide a means for more flexible sampling. Before examining the research into this area, some sampling concerns to be examined are given:

1. Determination of sampling density and distribution to sufficiently approximate the hemispherical distribution required for computing accurate form-factors, considering surface ra-

diative properties, and geometric relationships between sources and receivers.

2. Determination of a stopping criteria to halt traversal of a ray tree if ray tracing. The commonly used tolerance has been to use some sort of form factor contribution falloff as a threshold.

3. Determining how to reuse information for rays shot when computing a single form-factor, between form-factor computations, and between iterations if a progressive refinement technique is used. That is, what sort of of coherence can be exploited to avoid recomputation.

In the following discussion, sampling concerns will be examined from three different perspectives: 1) from the view of sufficient sampling from a source patch, 2) from the view of sufficient sampling at a receiver patch, and 3) from the view point of sufficient mutual sampling between both sources and receivers.

Source to Receiver Sampling

In the following methods, the initial assumptions are that samples are taken from the center of the source patch, and that the information obtained from these samples applies to the entire patch.

One of the earlier experiments uses ray casting through a uniform grid to compute form-factors ([Maxwell86]). Maxwell notes, that a large number of random numbers is required for good statistical results, if using a probability function for choosing directions to shoot rays in.

Malley ([Malley88]) attempts to reduce aliasing problems as a result of uniform sampling when casting rays. The hemisphere above a patch is sampled such that the density of rays is proportional to the cosine of the angle from the normal of the surface, since form-factor values drop off as a function of the cosine. Ray directions are jittered to anti-alias. Being oblivious to the geometry of the scene, the problem with this type of approach is that if nothing interesting is hit by rays tested at those angles, or if the directional emission does not match a cosine distribution, then rays may be wasted in directions of little importance.

[Sillion89] tries to capture specular effects using *extended form-factors*. A planar grid is used for ray tracing through, such that the grid consists of non-uniform cells, with a roughly constant associated delta form-factor per cell. Advantages include only requiring sampling through a single plane, and linear time retrieval of precalculated delta form-factors stored in Crow texture tables ([Crow84]) which contain summed area delta form-factors.

Visibility as opposed to intensity is used to determine adaptive sampling by using an idea based on *Warnock area subdivision* ([Foley90]). Rays are shot from the center of the source through grid points, distributing delta form-factors to ray trees produced. For each pair of rays along a grid edge the grid cell is subdivided and resampling occurs if the ray trees are different. A potential problem is that a very large amount of subdivision may be required in some cases to capture visibility gradients. To save time, rays are stored for reuse if subdivision occurs. The factors to weigh are storage cost versus recomputation cost, both of which are dependent on the grid resolution.

Common to these methods is that they do not consider the sample locations required on the receivers with respect to the source when sampling. Also, by sampling from a single point on the source there is the potential to undersample the source.

Receiver to Source Sampling

[Wallace89] addresses these problems, by sampling from the receiver to the source, providing better source sampling, and eliminating shooting rays that do not hit anything. Sampling from element vertices allows the usage of the exact geometry of surfaces, and the degree of sampling is adjustable based on the degree of accuracy desired, allowing a rough estimate of the number of rays required to be precomputed.

If uniform sampling of the source is used then aliasing effects appear, which are especially noticeable if the distance between source and a receiver is small, and the area of the source large. Alternatives included: a) jittering sampling positions, to substitute large scale noise for aliasing; b) filtering samples by taking weighted averages of neighbour vertices; and c) adaptively subdividing the source until the the radiosity impinging on a receiver vertex is less than some tolerance. This is not implemented, but is the preferred choice to sample the source sufficiently.

The form-factor formulation is based on an analytic differential area (receiver vertex) to parallel disc (source) approximation ([Siegel81]), with an associated degree of error when the source is not a parallel disc. From experimental results, undersampling occurs for sources and receivers at right angles to one another. This may require extensive subdivision to capture the resulting gradients.

Mutual Sampling

[Hanrahan91] takes a two-step approach to computing form-factors. Initial form-factor values are computed assuming complete visibility, which are then refined later on by computing form-factors with visibility. A jittered ray casting method based on [Thibault87] is used.

Hanrahan notes that form-factors are prone to error, and therefore need only be computed and stored if the error is within a given tolerance. The difference in mutual form-factors between two patches is suggested as an error measure. This measure restricts size, distance and orientation allowed between two patches, before a form-factor value is stored (*interaction* is allowed) between them. What results is that the number of interactions for a given patch is restricted to a constant amount with the total number of interactions proportional to the number of patches and elements. As previous methods did not restrict interactions, $O(n^2)$ interactions needed to be stored as opposed to $O(n)$ in Hanrahan's method. n is the number of patches in the scene.

Visibility need only be calculated as accurately as required for form-factors within the error tolerance, and roughly the same amount of work is done at each visibility test. Therefore the amount of visibility testing is roughly proportional to number of form-factors interactions. Note that if two patches are mutually visible or non-visible no further tests are required. It is noted that this method works best for scenes with few large initial polygons with high radiosity gradients.

As Hanrahan restricts the number of visibility and form-factor interactions by using an error measure, so may the amount of work for visibility testing also be restricted.

First, back-face culling ([Foley90]) with respect to the source and receiver planes may be used to cull away objects to restrict the number of ray intersection tests. As noted in [Haines92a], care must be taken to choose appropriate sample positions, as the case may be that the sample position on the source can not be seen by the receiver, but other parts of the source may still be seen. Haines and Wallace ([Wallace89]) also suggest using object intersection testing versus testing the mesh representing the object.

In [Wallace89], hierarchical bounding volumes are used to reduce the number of intersection tests per ray to a logarithmic amount with respect to the number of objects ([Weghorst84]). A "good" hierarchy is a one where the increase in volume between any two levels of the hierarchy is minimized in order to reduce the number of volumes that need to be checked per ray.

Haines ([Haines91a]) restricts the bounding volumes and objects to test by noting that the visibility between two patches is restricted to a finite volume defined by a **shaft** which is basically the convex hull of the bounding volumes around the two objects. All bounding volumes or objects outside this volume may be discarded by using back-face culling for each face of the shaft as demonstrated in *Figure 4.6*). The effective savings of this technique is dependent on the ratio of rays used per shaft, how tight the bounding volumes are, hence how tight the shaft is, the meshing of the scene, the definition of an object, and the geometry of the scene. Tightness means how closely a bounding volume matches the volume of space occupied by the object. This implementation is only good for opaque surfaces, and the best bounding volume selection and best definition of an object are still unknown.

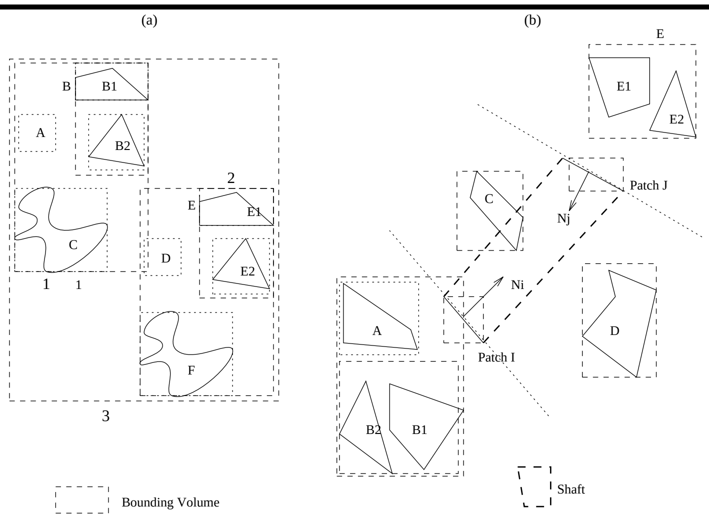

Figure 4.6: Coherence: a) Hierarchical bounding volumes are shown (in 2D). Volumes E1 and E2 are contained in E, volumes E, D and F are contained in volume 2, and so forth. b) Culling: All objects in volume E are culled by the planes that patch i and patch j lie on. Objects A, B1 and B2 are culled by traversing the bounding volume tree. Object D is culled by the planes of the shaft.

4.3.5 Form-Factor Storage Techniques

To conclude discussion on the first category of form-factors, the major steps that mark the progression in the amount of storage required for form-factors, will be examined.

In **full matrix** methods such as ([Goral84]), since all form-factors between all pairs of patches must be stored for radiosity computation, $O(n^2)$ storage is required at all times, n being the number of patches. In **iterative** storage methods, the storage cost is $O(n)$ per iteration of a progressive refinement algorithm ([Chen89]). If form-factors are to be stored for future use, the total storage for form-factors is still $O(n^2)$.

The number of patches may change during the execution of the program, and thus n may change in value in both approaches. Also, if form-factors are computed between entities other than patches (e.g. vertex to patch) then term n would represent the upperbound on the number of entities, though the magnitude of storage would remain the same.

Lastly, [Hanrahan91], only stores form-factors between patches if the associated error is less than some tolerance. Total form-factor storage is reduced to being linear to the number of patches (n), as only a constant number of form-factor values are stored per patch. The worst case is for parallel surfaces and the best case of $O(\sqrt{n})$ total storage is for perpendicular surfaces.

4.3.6 A Ray Tracing Approach

The second category of form-factors, will now be examined. In particular, research into using ray tracing to propagate energy as in a radiosity approach, will be looked at, where the aim of such research has been to add surface-to-surface indirect energy transport to standard ray tracing approaches. The basic difference between these methods and previous methods presented, is that energy is propagated with each sample (ray) shot, as opposed to a separate computation of form-factors before propagating energy. In general, the basic concepts are similar to the Monte Carlo method of simulation presented in *Chapter 3*, where a form-factor is roughly equivalent to the percentage of rays that reach some patch j from an original patch i. Emphasis will be on examining relevant points of interest, rather than attempting to cover algorithms in detail.

Before discussing any of the algorithms, a few relevant ray tracing ideas will be examined. Discussion of the algorithms will then follow.

Ideas From Ray Tracing

The first important idea is that of using an *importance* criteria to reduce the amount of sampling to perform. [Kajiya86] uses an adaptive hierarchical sampling scheme, concentrating samples in areas of high variance, where importance is based on the surface's reflection model. A huge number of directions still needs to be sampled if the reflection model is complex ([Sillion89]).

Ward ([Ward88]) reduces the amount of sampling required by concentrating on sampling specular or rapidly varying effects, and computing fewer samples for slowly varying effects. This is accomplished by computing new samples based on the density of previous samples. In addition, samples are made independent of the surface geometry, by storing samples in an independent data structure.

The last idea to examine is a common thread between radiosity and ray tracing as seen in ([Arvo86]). Arvo presents a technique called *light ray tracing (LRT)* where, for each emitter, energy propagates via one or more specular surfaces to a diffuse receiver. This is basically an iteration in a progressive refinement method. A major problem with this method was insufficient sampling for accurate reconstruction of radiosity gradients on receivers without undue cost.

Ray Tracing Algorithms

We now turn our attention to discussing the ray tracing algorithms, previously mentioned.

The first algorithm to examine was presented in [Shinya89]. Shinya uses *pencil tracing* ([Arvo89]) from light sources to capture caustics, focuses, and dispersion effects. Caustics and focuses are complex patterns of light concentrated on a surface due to refraction and reflection. Important ideas included sampling using a pyramid of rays from sources, shooting from source subdivision points to better sample area sources, and filtering of samples deposited as maps on receivers.

Watt ([Watt90]) used similar ideas in the form of beam tracing ([Heckbert84]), to also capture caustic effects. The prevalent ideas presented in this paper are: a) to shoot energy from emitters to vertices of specular objects first to allow for regions with rapidly changing effects to be sampled more, and b) to use beam tracing to deposit better sample distributions on receivers. Unfortunately, the method leaves samples that produce polygonalized gradients due to sampling at vertex positions on specular objects.

For better specular sampling, Shirley ([Shirley90]) performs adaptive sampling of specular objects. As in [Malley88], n random rays are shot in random directions from each source (non-uniform sampling) to "find" specular surfaces first, before sampling further in directions where specular surfaces are found. Shirley notes that if only close to ideal diffuse values need be captured using radiosity, then a "coarser" sampling of receivers may be used to compute these diffuse values. Overall, the technique is restricted to handling only near ideal or ideal diffuse or specular surfaces.

In the above methods, as with [Arvo86], there may be receiver sampling density / distribution problems since importance of sample positions on the receivers is not considered, and source undersampling may occur since rays are shot from the center of the original area of a source, or subdivided areas of the source without an appropriate source subdivision scheme being used.

Heckbert addresses some of these problems in [Heckbert90], which also uses light ray tracing to distribute energy in cosine distributions from sources. Important aspects discussed were: a) source subdivision and resampling on differences in visibility between rays; b) receiver subdivision based on energy density; c) original source sampling density based on stratified sampling such that the density is proportional to original radiosity; and d) original receiver subdivision based on projections with respect to an eye position.

An alternate adaptive source sample density is used in [Airey90]. [Airey89] and [Airey90] note, empirically at least, that at each iteration of a progressive refinement algorithm, the amount of unshot radiosity decreases as a negative exponent. This led to the idea of adapting the sampling density as a function of the amount of unshot radiosity at each step, in order to try to keep the radiosity shot out roughly constant per ray.

4.4 Capturing Gradients

As important as capturing geometric visibility are sampling techniques to capture energy gradients on surfaces. In the last section, the ray tracing view of "distributing" energy samples on receivers from sources was seen. From a radiosity view, how to "collect" energy samples will now be examined.

Basically, the use of subdivision to sample rapidly changing gradients on surfaces will be examined. Areas of concern include: a) subdivision of receivers, and subdivision of sources, and how well the two subdivision methods harmonize; b) selecting subdivision points, and what values to assign at these points; c) when to halt subdivision; and d) maintaining an even distribution of sample points on subdivision, by maintaining well-shaped surfaces and sufficient mesh density.

Most of the algorithms to be examined use bilinear subdivision, and assume monotonic change along gradients, when assigning values at subdivision points. Unfortunately, if a surface is initially insufficiently subdivided, gradients on the surface may be missed due to undersampling.

An additional concern is what subdivision should be permanently stored. Since radiosity samples at receivers are used to represent radiosity gradients, it makes sense to store this subdivision permanently, while since source subdivision is mainly performed to sufficiently sample the energy distribution of sources it either need only be temporary, or stored separately from receiver subdivision. "Receiver" and "source" labels only apply at a given time, as they may switch roles at different times. Conflicts may thus arise as how to treat a receiver patch at time t, when it becomes a source at some other time u. A general assumption is that a source is defined at a patch level, and not at an element level, which ignores energy distributions on the source.

The following discussion is broken up into two parts: First, consideration of capturing gradient information at receivers will be examined. Secondly, how to sufficiently sample sources will be looked at.

4.4.1 Receiver Gradients

The first usage of automatic adaptive subdivision is seen in [Cohen86]. Radiosity gradient information is captured by using bilinear subdivision in areas with high radiosity gradients. This avoids overmeshing in areas with little change, thus avoiding the associated increase in mesh storage cost. There are, however, a number of problems associated with this method. First of all, since interpolated values are used to identify gradients, and monotonic change is assumed along subdivision edges, gradients may be incorrectly identified, and / or missed. Also, if a gradient is not aligned along the subdivision grid, then bilinear subdivision may require many levels of subdivision before only reaching an approximation of this gradient. An example is shown in *Figure 4.3*. Lastly, since an ad hoc tolerance threshold is used to halt subdivision, possible over or undermeshing may result.

[Airey90] suggests to instead split along the radiosity gradient if possible. For quadrilaterals this would mean a split along the diagonal joining the two vertices with the smallest difference in radiosity values. If the difference is still too high, then quadrilaterals are split into four triangles.

As seen in *Figure 4.7*, a radiosity boundary may be missed by only using radiosity information ([Arvo91]). Additional subdivision based on a threshold difference of form-factor values between sample points of a receiver is proposed in [Campbell90] and [Arvo91]. Campbell's method has some additional points of note. First of all, receivers are split along their long axis to try to maintain well shaped surfaces. If splitting is performed, then subdivision is performed after each receiver is tested, versus after every iteration (all receivers tested) for faster image update. The method also has the advantage of not requiring visibility to be rechecked when receivers are subdivided since receivers have already been cut along visibility boundaries during visibility testing. Finally, the subdivision mesh created during visibility testing provides more accurate first approximations of radiosity boundaries on receivers than previous bilinear subdivision methods. Unfortunately, since cuts are permanent, sample positions may be generated throughout the environment that may not be along radiosity gradients.

As previously mentioned, a roughly equivalent numerical approach to Campbell's is used in Baum ([Baum91]), where splitting is based on pixel level occlusion computed using a depth-buffer. Splitting may be excessive, and gives less accurate boundaries, since bilinear subdivision is used. To avoid resampling, only the current source is reshot at new receiver sample points. To prevent over or undersampling, a minimum and maximum polygon edge size is used. An alternative minimum receiver size criteria of minimum reflectance threshold is given in [Nishita85].

4.4.2 Source Sampling Techniques

We now turn our attention to sampling energy distributions on sources properly.

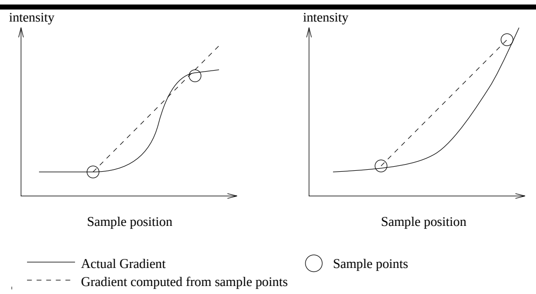

Figure 4.7: Radiosity boundary is missed in example on left, though caught in the example on the right. (After [Arvo91])

The first source sampling method to examine, in [Nishita85], handled source sampling by sampling from receiver vertices to source vertices. Since this method is not adaptive, sources are in effect simulated as uniformly distributed surface points, regardless of the energy distribution across the surface.

To better capture these distributions an adaptive technique is required. The first adaptive subdivision technique ([Chen89]) came about as a result of the requirement to maintain form-factor relationships. That is, due to violations in the reciprocity assumption, the value of a form-factor F_{j-i}, where j and i are the receiver and source respectively, could become greater than one, which is a physical impossibility. For such cases, bilinear subdivision was used to bilinearly subdivide the source, and form-factors were recomputed.

To sufficiently sample sources, Wallace ([Wallace89]) adaptively bilinearly subdivides sources when the radiosity received at a receiver vertex is greater than some maximum energy threshold. Even if the sources have very strong radiosity, or a steep radiosity gradient across the surface, they should not be undersampled, nor should small area sources be missed. Wallace suggests that an efficient way to compute the subdivision would be to prestore the source subdivision, perhaps using a quadtree. When sampling, only tree traversal down to the appropriate level is necessary to find the sample distribution required.

As with receiver subdivision, a size constraint may be imposed for sources. [Campbell90], along the lines of [Wallace89], subdivides sources that are large or have strong radiosity with respect to the distance to a receiver, but adds the condition of halting if source elements become too small. To also avoid excessive source meshing, a condition of stopping after $\frac{1}{2}$ of the radiosity has been shot from the source, is used. This contribution is measured from the center of the current receiver.

Finally, if the solid angle subtended by the emitter is too small, Campbell suggests approximating it as point source.

In [Sillion91], the reflectance functions for sources and receivers are used to weight samples, and notes that energy distributions stored with elements of the source patch should be used. To this end, Hanrahan ([Hanrahan91]) adaptively subdivides based on differences in values: $B_i F_{i-j}$ and $B_j F_{j-i}$, where i and j are the current source and receiver, to consider both source and receiver radiosity distributions. This method is called *BF-refinement*.

Heckbert ([Heckbert90]) attempts to avoid meshing objects before results are known, by using textures (stored as quadtrees) mapped to the original geometry of objects. Sources are bilinearly subdivided *before* sampling if the power per quadrant is greater than some tolerance. On subdivision, power is equally redistributed to each new quadrant created, and neighbours one level up in the tree are also split, so as to maintain gradual energy gradients. To maintain uniform energy distributions, both minimum and maximum subdivision levels are proposed.

4.5 Radiosity Solutions

With techniques for computing direct energy transport, consideration is now given to solving the global illumination problem using radiosity solutions.

For a "standard" approach, (as shown in *Chapter 1* and [Goral84]), following the computation of form-factors, a set of n linear equations that compute the radiosity at each of the n patches may be solved by direct or iterative matrix methods. The computation of each equation is akin to "gathering" energy at a patch from all other patches. Algorithms using this approach will be labeled *full matrix* or *FM* algorithms. Algorithms that solve the set of equations in a iterative manner, as introduced in [Chen89], will be called *progressive refinement* or *PR* algorithms.

Both FM and PR solutions are considered to be *single pass* algorithms, as solutions are computed using a single pass of an algorithm. *Multi-pass* solutions use more than one algorithm, in sequential order. Accordingly, solutions will be presented in two parts: 1) single pass solutions, and 2) multi-pass solutions. Where applicable, the manner in which transport of energy is achieved, the illumination model(s) used, and the resulting global illumination effects, will be examined. First, a list of terms and symbols to be used is given in *Table 4.1*. As an example, "$LS_g^* D_i$" will mean light to zero or more general specular surfaces to ideal diffuse surface transport. Following the presentation of these solutions, the order in which energy is propagated, physical accuracy and other error measurements, and radiosity storage techniques will be discussed.

Symbol	Description
E	eye : current viewing position
L	light : surface with non-zero original emission
S	specular : surface with specular component to local illumination model
D	diffuse : surface with diffuse component to local illumination model
$\vert$	or
()	grouping delimiter
[]	optional
i	ideal properties
p	planar property
g	general property
*	zero or more
+	one or more
Term	**Description**
FM	full-matrix method (see [Goral84])
PR	progressive refinement method (see [Chen89])
HC	hemicube method (see [Cohen85])
RT	ray tracing from eye to the scene (see [Whitted80a])
DRT	Distributed ray tracing (see [Cook84])
MCPT	Monte Carlo path tracing (see [Kajiya86])
BT	beam tracing (see [Heckbert84])
BV	bounding volume (see [Weghorst84])
LRT	light ray tracing or backward ray tracing (see [Arvo86])

Table 4.1: Transport terms

4.5.1 Single Pass Methods

The first set of methods to consider are those that compute radiosity solutions using a single pass of a radiosity computation. First FM solutions, then PR solutions will be covered.

Full matrix algorithms

To solve the system of previously mentioned equations, Gaussian elimination with partial pivoting was originally used ([Goral84]). Since properties of the matrix containing reflectance and form-factor terms is not exploited, an $O(n^3)$ time complexity results, n being the number of patches in the scene. By taking advantage of the strictly diagonally dominant nature of this matrix, Cohen ([Cohen85]) is able to use the Gauss-Seidel iterative method, taking the emission of the patches as the initial guess. Time is reduced to $O(n^2)$.

By examining form-factor relationships, further reduction is realized in ([Cohen86]). For previous methods, a fixed sized matrix resulted from a fixed number of patches. If this number increases the time and storage also increases. Cohen allowed the size of the matrix to be independent of subdivision, by not storing element form-factors on subdivision, but instead taking area weighted averages of element values as patch values. This gives a $O(ne)$ solution, where n is the number of patches, and e the number of elements.

These original methods achieved diffuse to diffuse transport, with the assumption of opaque surfaces with ideal diffuse reflectance and / or emission. There have been a number of single pass extensions to include non-diffuse effects. Physically-based effects are simulated in: [Immel86], [Rush86], [Rush87] and [Rush90a]. Immel ([Immel86]) allowed arbitrary reflectance functions, and maintained conservation of energy, which is often overlooked. Testing with Phong ([Phong75]) and ideal specular reflectance functions, computation time was unfortunately in the order of VAX 11/780 months to compute a single image. [Rush87] added participating media with wavelength-independent isotropic emission, absorption, and scattering, and opaque surfaces with Lambertian reflection, and emission. [Rush86] and [Rush90a] allowed translucency for surfaces with the same characteristics, but doubles the number of hemicubes required per translucent surface.

Implementation of empirically based effects have also resulted through the use of texture mapping. Texture mapping was first used in radiosity algorithms in [Cohen86]. The basic problem with the method used is the assumption of constant reflectance and radiosity per map during the radiosity computation. To handle the "flatness" problem with texture mapping, bump mapping was used in [ChenH90] by jittering the normals used for hemicube calculations to simulate rough surfaces. A basic problem with this method is the sparseness and uniformity of the bumps.

In summary, general drawbacks of all full matrix solutions are the computational and storage costs for the matrix, and that solutions cannot be adaptively refined ([Bergman85]).

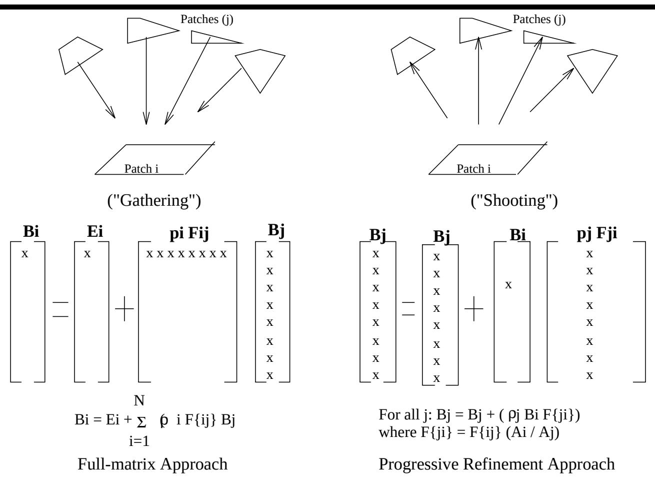

Figure 4.8: a) Full matrix method: All equations for B_i must be solved simultaneously. b) Progressive refinement method: Solutions for B_j may be updated iteratively.

Progressive Refinement Algorithms

The original *progressive refinement algorithm* ([Chen89]) is an iterative method, using ideas very similar to those of light ray tracing ([Arvo86]), and the Atherton-Weiler visible surface algorithm ([Foley90]), to simulate ideal diffuse to ideal diffuse transport (D_i^+E). The form-factor computation from sources to receivers is the same as for a FM method, but the reciprocity property is used to compute form-factors from receivers to the source, resulting in storage linear to the number of receivers being required per iteration. Once form-factors are computed, the radiosity from the current source to all applicable receivers can then be found. This process is akin to "shooting energy" to the receivers.

For example, assuming that there are n patches in a scene, and patch i is the current "shooter" with B_i^{unshot} radiosity left to propagate. All patches B_j for $j = 1$ to n may have their current radiosity $B_j^{current}$ updated to B_j^{new} as follows, due to patch i propagating energy to them:

$$\begin{bmatrix} B_1 \\ B_2 \\ \cdot \\ \cdot \\ \cdot \\ B_n \end{bmatrix}^{new} = \begin{bmatrix} B_1 \\ B_2 \\ \cdot \\ \cdot \\ \cdot \\ B_n \end{bmatrix}^{current} + B_i^{unshot} \begin{bmatrix} \rho_1 F_{1-i} \\ \rho_2 F_{2-i} \\ \cdot \\ \cdot \\ \cdot \\ \rho_n F_{n-i} \end{bmatrix} \tag{4.1}$$

ρ_j is the reflectance of patch j, and F_{j-i} is the form factor from j to i and is computed using the reciprocity rule as follows:

$$F_{j-i} = F_{i-j} \frac{A_i}{A_j} \tag{4.2}$$

where A_i is the area of patch i, and A_j is the area of patch j. F_{i-j} is computed using the hemicube method for patch i.

By iteratively setting different patches to be the source or the *shooting patch*, the radiosity for other patches can be incrementally updated until a sufficient number of patches have "shot" their energy.

To produce faster initial images, an *ambient term* based on form-factor and reflectance approximations, and the current radiosity is used, which decreases as the true solution is computed.

Due to reduced storage cost in this method, more complex scenes may be rendered, with "faster" initial solutions, and smooth image refinement to a complete solution. As such, there are many single pass methods which use this progressive refinement algorithm ([Cohen88], [Malley88], [Baum89], [Airey90], [Campbell90], [ChenH90], [Chen90], [Recker90], [Rush90a], [Rush90b], [Wang90], [Baum91], [Dorsey91]).

4.5.2 Multi-pass Algorithms

The need for new approaches arises mainly due to excessive storage and computation time to compute specular effects (e.g. [Immel86]). The main reason for inefficiencies is that mechanisms, best suited for capturing diffuse transport, are extended to capture specular effects. In the same manner, ray tracing approaches also become very expensive when extended to capture diffuse transport (e.g. [Kajiya86]). A natural progression is a hybrid radiosity-ray tracing approach, to take advantage of the best aspects of each.

Relevant characteristics of radiosity methods are view-independent diffuse interreflections, physically based solutions (i.e. no empirical ambient term), purely geometric form-factors (no need to recompute on surface property changes), and ability to capture transport from sources to diffuse receivers well. Important characteristics of ray tracing methods are consideration of viewing position, and efficient computation of specular transport.

The first hybrid two pass method ([Wallace87]) performs a FM radiosity computation using translucency and planar specular reflectance methods from [Rush90a], for $D_i^+ S_p D_i^+$ transport. A ray tracing approximation second pass is used to capture $D_i^* S^* E$ transport. Note that an incomplete solution results from using a post-process or a pass to perform specular transport to the direction of the viewer only. This is because only effects relevant to the current viewing position are considered. This type of computation is more efficient, though, than computing specular transport to all directions in the scene. Also, the same transport mechanisms and illumination model should be used in all passes, and all transport dependencies be considered otherwise the incorrect results are computed. For Wallace's algorithm, due to oversimplifying transport into 4 mechanisms (D_i to S_i, S_i to D_i, S_i to S_i, and D_i to D_i transport), higher order effects that occur when energy arrives at a surface via a path involving several interactions with several surfaces where any of the mechanisms may be involved in any order are missed.

[Shao88] attempts to address the high cost of using the image method, and to achieve more accurate diffuse transport by considering diffuse and specular impinging energy on a surface simultaneously, since outgoing energy is dependent on both of these terms. A first pass uses FM radiosity and stores visibility for non-diffuse surfaces found. The second pass recalculates (re-samples) "form-factors" for non-diffuse surfaces using FM radiosity. A ray tracing post process is used to find specular highlights. The main drawback of this method is that "form-factors" are not purely geometric, requiring recomputation on spectral and spatial property changes as well as surface geometry changes.

[Sillion89] uses a two-pass method of FM radiosity followed by a ray tracing pass to achieve $D_i^+ S_i^* D_i^+$ and $D_i^+ S_I^* E$ transport, ideal refraction, and non-planar specular effects. The first pass uses extended form-factors to include non-planar specular effects. An eye pass interpolates radiosity values from diffuse surfaces instead of using shadow rays ([Arvo89]). For specular surfaces, shadow rays still must be shot, since no energy is stored on specular surfaces. Since samples are only kept per patch, source to specular to diffuse patch contributions are not captured well. Therefore the method is poor at capturing effects such as caustics.

[Watt90] uses roughly the same approach as in [Wallace87], except with a different method for diffuse exchange (beam tracing). A first pass of LRT computes $LS_i D_i$ transport and a second ray tracing pass captures $D_i S_i^* E$ transport. The major advantages of this method are better specular surface sampling, and maintaining information about directions to specular surfaces that were visited in the transport of energy to diffuse surfaces. This information can be used in the second pass, for better diffuse reconstruction. This brings up another problem with most multi-pass methods, that of reuse. Many either never mention how or if information common to different passes may be used, or exploit this information very little. Especially relevant is the reuse of ray information, since many use ray tracing in more than one pass.

To take into account viewing position, [Heckbert90] uses a first pass of ray tracing from the eye to ideal diffuse surfaces to obtain a mapping from world to screen space to determine minimum diffuse sampling densities. The other two passes are LRT for point light sources, and RT passes with only ideal diffuse and specular reflection considered. Due to the histogram storage method used to store energy at surfaces, "blocky" gradients are seen in images produced.

[Shirley90] made a distinction between diffuse to diffuse, and light to diffuse transport, using different mechanisms for each. A three pass solution of LRT, a FM radiosity for indirect energy transport between non-emitters, and a DRT pass to the eye that includes shadow ray testing to compute direct illumination at diffuse surfaces, is given. This method seems to be one of the most compete to date, with respect to transport mechanisms as it achieves $LS_g^* D_i$, $D_i^+ S_g^* D_i^+$, $(L^* D_i | S_g^*)E$ transport. A proposed guess for time complexity for using MCPT is $O(n)$, n being the number of patches ([Shirley90], [Airey90]).

[Chen91] notes that due to the ordering of the passes, diffuse interreflection effects are found before computing strong specular effects, textures, etc in the ray tracing pass to the eye. As well, the method is suited for only near ideal diffuse or specular surfaces. Chen reorders the passes in a three pass method. The first pass is a PR pass using extended form-factors ([Sillion89]) to obtain diffuse storage. Next is a "high frequency" pass of LRT to produce caustics or illumination maps, and MCPT with coarse sampling to capture effects to the eye. A final "low frequency" pass of MCPT or DRT to the eye is used to refine the images. A second shortcome of Shirley's algorithm ([Shirley90]) was the fact that only self-emitters were considered to be "light sources". To take into account "secondary light sources", such as highly specular surfaces visible to lights, surfaces were reclassified after the PR pass.

When storing impinging energy samples on diffuse surfaces or propagating energy most algorithms assume ideal diffuse properties, resulting in the inability to capture directional energy distributions. [Sillion91] stores general reflectance functions ([He91]) during a PR radiosity pass, and computes ideal specular effects using the image method as applied to a PR algorithm. That is, instead of reflecting the scene to produce "virtual worlds", only the current shooter is reflected. A ray tracing pass is used for rendering. Directional emissivity of surfaces is included in [Dorsey91] by using spatial weighting functions when computing outgoing radiosity.

4.5.3 Shooting Order Priority

Regardless of the means to propagate energy, some sort of propagation order must be imposed. The order chosen will affect the the manner in which a scene will be lit, and thus the speed at which parts of a scene will appear with their final illumination values in an image.

[Immel86] proposes to propagate from emitters first, since they have the most energy to propagate to the scene. This idea is quite similar to Arvo's idea of shooting from lights ([Arvo86]). The first shooting order implemented for PR radiosity algorithms ([Chen89]) sorts by how much energy has yet to be propagated from a patch. Different priorities will be discussed when considering solutions for dynamic environments.

4.5.4 Convergence and Error Measurements

When propagating energy, knowledge of when a "correct" solution has been reached is required, thus requiring some sort of halting criteria and error measure. Many of the criteria given in research has been based on experiments on specific computer models, and may not have any correlation to physically measured results on a real world analog. Currently testing has been restricted to ideal diffuse environments ([Meyer86]).

For full-matrix methods, convergence to a correct solution is assumed if the degree of error in the matrix solution is less than some tolerance. [Immel86] adds the criterium of convergence if the difference between successive radiosity values stored is below a tolerance.

For PR methods, Chen ([Chen89]) uses the criteria of halting when the percentage of the total energy in the scene left to propagate is less than a tolerance. An RMS error of radiosities for all elements in the scene is used to measure the rate of convergence with respect to the final PR solution: $Error_{RMS} = \sqrt{\frac{\sum_{i=1}^{e}(B_i^* - B_i)^2 A_i}{\sum_{i=1}^{e} A_i}}$ where e is the number of elements, B_i is the current radiosity, B_i^* is the final radiosity, and A the area of an element. Algorithms using planar grids to approximate a hemisphere, either lose energy forever ([Wang90]), or wait until the amount of energy lost is greater than the unshot energy of the surface ([Recker90]). Experimental results given show that the number of iterations until convergence is actually longer than PR methods not using a planar grid, since there is the need for extra "lost energy" propagations.

From experimental results given in [Airey89], [Baum89] and [Greenberg91], after roughly 100 iterations, the rate of convergence to within a given error (roughly 20 percent) of a solution found using a FM method, then converges slower than with a FM method. Unfortunately, accuracy is only measured with respect the FM solution. [Baum89] also uses a FM solution to compute accuracy using the error computation: $|B_{FM} - B_{approx}|_2 / |B_{FM}|_2$. $||_2$ is the square root of the squared radiosity over all colours and all emitters, subscript FM is for full-matrix and $approx$ is for their approximation. As mentioned, the solution is within 4 percent of a sample FM solution.

[Haines92a] notes that using the ratio of total unshot energy to total shot energy may give a high initial convergence rate which may be very misleading, as the time to converge to a final result may actually be slower. For example, imagine surfaces with high radiosity values that face nothing. Shooting from them will give a high convergence rate, but the solution has not actually converged. In general, it is noted that the rate of convergence is highly dependent on the nature of the scene, and it is still unknown what is the best general criteria.

Most methods using Monte Carlo, or other distributed ray tracing algorithms to compute diffuse effects have not been measured in relation to other methods, and are still considered intractable ([Greenberg91]).

Qualitatively, after about 100 iterations of a PR method, differences with respect to a converged result are said to be visually indistinguishable ([Chen89], [Cohen88], [Greenberg91]). Haines ([Haines92a]) also notes that for the illusion of realism it is not necessary to have totally accurate colour bleeding or sharp shadows, as commonly seen in ray traced scenes. In fact, sharp shadows may be interpreted as ambiguous "cuts" or "folds" in the scene.

4.5.5 Radiosity Storage Techniques

Having computed radiosity values, a storage mechanism is required. Factors to consider include storage cost, accurately in capturing gradients and recovering values for rendering, and what surface properties and illumination assumptions are used. Our discussion is split into two parts, based on two general structures used: polygonal and map storage.

Explicit Storage in Scene Geometry

The first type of storage technique stores radiosity values at exact geometry points given in a polygonal model. The two main choices have been per polygon, or per vertex.

The first choice assumes uniform values per polygon (e.g [Goral84] [Chen89] [Cohen88]). The second choice stores values directly at vertices of a polygon allowing variations in gradients across a polygon (e.g [Nishita85] [Wallace87] [Wallace89] [Sillion91]). [Wallace89] notes that multiple values may be stored at a vertex position if the vertex is shared by more than one polygon. Sillion ([Sillion91]) stores direction energy distributions in the form of splines. Advantages of the representation include the fact that transformations of distributions are simple, continuous representation of distributions, and that storage does not grow as samples are accumulated, as with previous methods.

General problems with these techniques are that polygonization is used to store samples, resulting in complications for meshing models, and rendering problems. The latter problems will be discussed in sections forthcoming.

Mapping to Scene Geometry

The second type of storage is in texture maps ([Foley90]), where values are stored at given points in a map associated with objects in a scene, rather then directly at surface mesh points.

Arvo's idea of storing textures per polygon is used by [Shirley90] and therefore suffers the same sampling problems due to storage in a regular grid ([Arvo86]). Maps are also used by [Watt90], but in the form of a set of *surface detail polygons* ([Foley90]) for better sampling densities. Final values are found by summing the values in each map present at a given location on a polygon. To avoid jagged maps on receivers, Shinya ([Shinya89]) filters the sample distributions deposited on receivers.

Rather than storing textures in association with the polygonal breakdown, Heckbert ([Heckbert90]) uses maps based on the original geometry of an object to allow smoother interpolation and independent storage. Bilinear subdivision is used to adaptively sample. There may still be problems with discontinuities within or between maps, and mapping textures to complex scenes.

For these methods, a very high degree of sampling may be required to sufficiently represent the diffuse radiosity gradients, which may also be polygonized.

4.6 Rendering

In order to be able to view the results of a simulation, the environment must be rendered. Two basic approaches employed to render the scene will be examined: *Gouraud shaded polygon scan conversion* (Gouraud shading), and *ray tracing*. Gouraud shading only renders the scene, without performing any visible surface determination with respect to the viewer, while ray tracing computes visibility as well as rendering.

Gouraud shading ([Gour71]) basically requires a set of intensity values given at vertices on a polygonized object. An image is formed by projecting objects onto an image plane. To obtain all values between a given set of vertices linear interpolation is performed. A basic ray tracing algorithm requires sampling the scene through a regularly meshed image plane. Vectors or "rays" are formed from the viewer through each grid cell. For the closest object that intersects the ray, the intensity is computed for the cell using the illumination model at the intersection point. Further rays may be sent out in specified directions according to the illumination model to capture effects such as reflection, refraction, and shadows.

4.6.1 Gouraud shading

For most algorithms that polygonalize objects in the scene, Gouraud shading has been used to render the objects ([Goral84] [Cohen85] [Cohen86] [Rush87] [Cohen88] [Chen89] [Rush90a] [Rush90b] etc). The main advantage for using this algorithm is it's speed and simplicity, as most of todays graphics workstations can render tens of thousands of Gouraud shaded polygons per second using algorithms incorporated into the hardware. For applications such as walk-throughs, and simulation of dynamic scenes, where real-time interactive feedback is desired, this method is preferable, Speed is also affected by the fact that Gouraud shading is used only to render the objects, and not perform any energy transport calculations. Thus for every image to draw, only a rendering operation is required, as opposed to ray tracing algorithms which additionally compute energy transport.

There are a number of problems with rendering a polygonized model though, as reconstruction of shading gradients is highly dependent on how well the polygonalization has captured the radiosity gradients on surfaces. Many problems result from the fact that a radiosity algorithm can produce large variations in radiosity within a polygon, which is rare in most previous uses of hidden surface rendering.

Mesh Density and discontinuities

To capture rapidly changing effects, a very fine mesh can be used, but this increases storage and computation cost, Though the use of adaptive subdivision ([Cohen86]) addressed this problem, it produces mesh discontinuities.

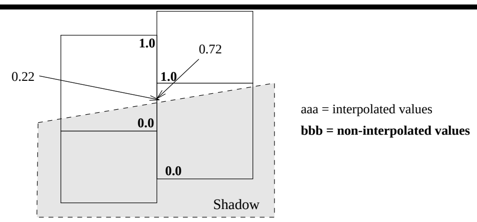

Figure 4.9: Shading discontinuity seen due to T-vertex. Polygon on left has value of 0.22 at shadow boundary, while polygon on right has value of 0.72

When interpolating values first order discontinuities may appear due to T-vertices (*Figure 4.9*), and light or shadow leakage. The basic solution given for T-vertices is to add these vertices to the polygons that do not share it originally. [Baum91] performs this when adaptively subdividing by anchouring, while [Haines92a] adds vertex points before rendering, but does not resample. Light / shadow leaks have been handled by splitting intersecting surfaces along lines of intersection ([Segal88] [Baum91]) and non-bilinear subdivision ([Campbell90]). Gradual mesh density changes has been addressed in [Heckbert90] and [Baum91] by not allowing mesh densities to differ by more than one level.

Rendering shape

When rendering polygons, two choices for shape have been examined: quadrilaterals, and triangles. Quadrilaterals are rotation variant (i.e. if rotated, shading changes), while triangles are rotation invariant but the appearance may differ depending on which diagonal of a quadrilateral is used for splitting. For example *Figure 4.10* shows the connection of the two white corners, and two black corners of a quadrilateral. Connecting the corners with the smallest difference has been suggested ([Airey90], [Haines92a]), as well as using a perspective warp function ([Wolberg90]), but this function is not supported by most hardware. This function may be approximated by splitting a polygon into a mesh of polygons, and interpolating new samples. This is slower, but gives better quality. There is currently no general solution for shading arbitrarily shaped polygons (e.g. with holes).

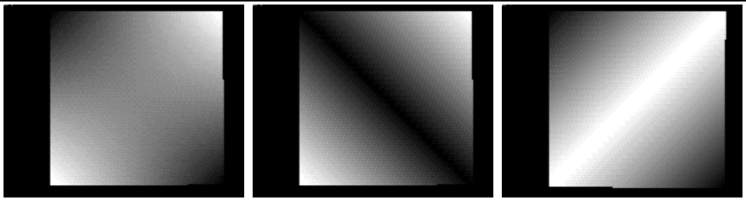

Figure 4.10: Gouraud shading: Left-most polygon is quadrilateral shading, middle is split along one diagonal, and right is split along other diagonal.

Shading Reconstruction

To shade, a set of shading values is required, thus a means to reconstruct the shading gradient is required. Unfortunately, information may be lost while reconstructing. For example, [Goral84] computes vertex values by interpolating from patch values. The problem of interpolating from insufficient information is quite obvious when rapidly varying gradients are to be captured. Images from [Immel86] show "washed-out" specular effects, for example. To avoid patch interpolation, values are stored directly at vertex positions, and averaging is used if more than one value is stored at a shared vertex ([Wallace89] [Arvo91]).

Haines ([Haines92a]) notes that if the rate of change between two polygons is different, Mach banding occurs ([Hall89]). This requires better shading reconstruction instead of bilinear interpolation. Cubic interpolation using nearest neighbour information is suggested by Haines, which is good for uniform sample distributions, but unsuitable if adaptive subdivision is used. [Chen91] suggests using a cone filter to reconstruct gradient values.

Side-Effects

Using Gouraud shading restricts choices for other techniques that can be used. This mainly results from requiring polygonal models, such that sampling and storage must be performed at fixed points on polygonized surfaces when computing form-factor and radiosity values. Additional overhead is required to support the techniques to maintain geometric constraints so that rendering artifacts do not appear ([Baum91] [Haines92a]).

4.6.2 Ray Tracing

In recent hybrid algorithms, the computation of rapidly changing effects has been separated from that for slowly changing effects (e.g. [Wallace87] [Wallace89] [Shirley90] [Chen91]). Ray tracing algorithms in general have been the preferred choice for capturing rapidly varying effects when rendering with respect to a viewing position, and during computation of the radiosity simulation.

Using a ray tracing method for simulation and rendering has allowed for the use of separate data structures for radiosity simulation and for rendering. This allows for more flexibility in the types of representations that may used for modeling, and sampling methods used for computing form-factor and radiosity values, as less polygonization is required.

The complexity of the ray tracing algorithm will differ depending on the types of effects to capture, and the level of detail desired. The complexity may range from using a simple ray caster to using complex Monte Carlo methods. Note that the algorithm used for simulation should match the one used for rendering, otherwise the results will be incorrect. A major hurdle has been to try and capture general energy distribution functions in an efficient manner (due to their high computational cost). In general some sort of distributed ray tracing approximation method has been used ([Wallace87], [Wallace89], [Shirley90], [Chen91]).

The effects of the simulation are usually stored at diffuse surfaces either in a polygonal model or using textures. To incorporate these values when rendering, a mechanism is required to retrieve them at ray-object intersection points. In general some sort of interpolation or filtering is used to retrieve values, which when weighted by the reflectance at the intersection point, is used as the "diffuse component", instead of casting shadow rays (e.g [Sillion89] [Shirley90]).

Care should be taken, though, when using using multiple representations to compute shading values, as errors may be introduced due to the differences in assumptions between representations. For example, if a polygonized model is used to store radiosity, a value at an intersection point may include a ray traced value using the real normal at the point on the surface along with a interpolated mesh value, which may ignore the geometry.

As this method is view-dependent, in terms of image production, regardless of the ray tracing algorithm used, all pixels of the image plane must be computed before a complete view of the scene is possible. Recomputation is also required at every frame, if the viewer moves. As previously mentioned, ray tracing performs an energy transport computation which has not been standardized in hardware implementations, and as such is slower than using hardware assisted Gouraud shading. Due to these speed and view dependencies, its usage has not been presented in real-time interactive applications for radiosity.

4.7 Changing Environments and Image Refinement

If the scene changes or the viewpoint is moved, then a new set of issues, need be considered. Environments where the geometry or properties of the objects may change will be called *dynamic environments*, otherwise they will be called *static environments*. *Walk-throughs* involve moving the viewing position with respect to the environment. In the following sections, issues for dynamic environments, and image refinement will be discussed.

4.7.1 Dynamic Environments

In dynamic environments, surface property and geometry changes may require recomputation of surface radiosities for one or more surfaces. The most expensive changes are geometrical ones, as form-factors, which are the most computationally expensive part of a radiosity simulation, need to be recomputed. The basic strategy to minimize recomputation time has been to only recompute values directly affected by any change, by exploiting temporal coherence (coherence over time).

Nishita and Nakamae ([Nishita85]) proposes to reduce the amount of recomputation of form-factors on movement of objects by performing a *shadow pass* to determine visibility. On any movement, they propose that less form-factors need be recomputed since not all geometric relations between objects will change.

[Baum86] reduces computation per frame by using a construct called a *back-buffer*. *Object coherence* is exploited by taking advantage of geometry that remains constant over time. Temporal coherence, is exploited by using predefined paths of object motion, and avoiding unnecessary computation by culling polygons that can never be seen from the objects path. Basically, visibility for each polygon that does not move in the scene is stored in buffers (back-buffers), taking into account the positions of visible dynamic objects. When updating a frame no form-factor computations are performed between static and dynamic objects, only between dynamic objects. Disadvantages of this method are that predefined sweep volumes are required for dynamic objects, plus preprocess and storage time for visibility calculations for all surfaces that may be occluded by dynamic ones. Additionally, only environments with planar ideal diffuse polygons are allowed.

[Sillion89] handled surface property changes by computing the effect of removing previous energy propagated to a surface, and reshooting the current energy for that surface. A single *incremental radiosity* value is computed to update surfaces effected. Propagation of energy is accomplished by shooting this incremental radiosity.

[Chen90] contains a more complete solution to handle moving objects, as well as using ideas from [Sillion89] for property changes. Chen does not attempt to prestore visibility in image space ([Baum86]), or in object space ([Buckalew89]), which both require a large amount of preprocessing storage ($O(n^2)$ for [Buckalew89], n = number of surfaces). Instead, interactive computation with non-predetermined changes are desired.

The first idea introduced was an *incremental form-factor* computation for geometric changes. An incremental form-factor between every patch affected by the patches on a moving object is computed based on old and new form-factors between these patches. Since visibility between each pair of patches, or between the object and patches only affects a small part of the scene, *modeling coherence* can be used to trivially discount objects. If ray tracing is used for computing form-factors, hierarchical bounding volumes can be used to clip away objects, or a restricted size hemicube can also be used if form-factors are computed using the hemicube method. The idea of reusing visibility or form-factor computation by caching the most frequently used form-factors is also suggested. To track geometry changes, a queue is used, with two basic actions being allowed: addition and deletion of objects. The queue can be used to find unchanged and changed patches, via the history of changes, and allows multiple changes to be compressed to save time. The current implementation works for ideal diffuse environments. Without space subdivision, a modified hemicube is used to handle geometry changes, and only interactive changes are allowed. Some drawbacks include no mechanisms for handling adaptive subdivision, and the storage and time for handling the queue for every patch in the scene.

With many changes occurring asynchronously, there is the problem of maintaining an up-to-date energy distribution in the scene. The criteria for choosing shooting patch order has thus been enhanced. Sillion [Sillion89] uses the *absolute energy* of a patch, while Chen [Chen90] adds the additional criterium of a form-factor approximation (the area of the patch with respect to the area of the world), and how much of the queue for the patch has been processed (i.e. the most up to date one). The reflectance of all patches the candidate is shooting to is ignored, by assuming constant reflectance. [George90] gives a shooting order criteria based on a form-factor approximation of unoccluded project area from the center of the patch. This method is flawed if a) the distance between the source and receiver is small, b) the projected area of the new object is roughly the same in all directions or c) there is occlusion. In general, this area has only been examined lightly with no guarantee that the errors produced by changes will reduce rapidly.

4.7.2 Image Refinement / Walk-throughs

Both during, and after our radiosity computations, some form of adaptive image feedback is desirable. Refining images has generally taken place when the viewer is stationary, with approximations used when the viewing position is changing. How images are refined and speed of computation will be examined.

Refinement of Image

The first attempt at providing refinement for radiosity solutions was in [Chen89]. Images are refined after a shooting patch has propagated it's energy into the scene. A faster method, proposed in [Campbell90], is to update on every receiver patch propagated to.

A proposal put forth in ([Wallace87]), is to only compute the diffuse components of energy transport while the viewer is moving through the scene. Only when the viewer is stationary is a post-process of ray-tracing performed to capture specular effects. There may be the problem of discontinuities in the effects seen in the image, when stopping and starting, and incorrect results.

[Airey90] performs post-processing when stationary, by retesselating patches along intensity gradients and recomputing radiosity values for new patches. Speed in recomputation of visibility was addressed by precomputation of model space subdivision. Speed on change of surface properties was addressed by precomputing the radiosity for 20 different types of lights sources.

[Baum90] makes use of viewing parameters as hints as to which parts of the scene should have their illumination calculations accelerated.

Approximations for Faster Computation

In order to facilitate faster computation, several approximation techniques have been presented. [Chen90] uses a scene property dependent *ambient term* to give better initial results (i.e. closer to the final image). [Sillion90] uses adaptive hardware lighting for early results, for similar reasons. Both [Sillion89] and [Recker90] use single plane hemisphere approximations for faster form-factor computation. [Sillion89] also uses the technique of depth-buffering mirror images onto a frame buffer for fast planar mirror form-factor approximations. Finally, [Sillion90] uses *negative light* shooting to allow interactive light changes, and includes the ability to alter surface meshes interactively. Also presented was a set of interactive movements for the user to walk-through scenes.

4.8 Parallelization of Radiosity

In general there has not been much research into parallelizing solutions. Two radiosity solution approaches, both of which use a hemicube form-factor method or some variation of it, and a VLSI solution to computing diffuse form-factors will be presented. Note that these approaches assume ideal diffuse surfaces, and that adaptive subdivision is not included.

4.8.1 Radiosity Solutions

Chen ([Chen89]) uses a local area network with a single master and a number of slave processors to compute a progressive refinement solution. Each slave computes a form-factor for a given shooting patch at a time using a hemicube method utilizing hardware depth-buffers. The master coordinates the radiosity solution, and display hardware, and assigns to each slave a patch to compute the form-factor for, such that if there are n slaves, then n form-factors for n source patches may be computed at any given time. Double buffering ([Foley90]), is used so that rendering of images need not be synchronized as old images are simply overwritten with new images. Problems with this solution are that a full geometry description of the scene is required at every slave, and communications and server bottlenecks reducing the effective speed increase due to parallelizing. The result is a less than linear increase in speed on a small number of slaves (less than 10), before bottlenecks are encountered.

This method has been used by various others including [Airey90], [George90], [Recker90], and [Sillion90],

An alternative solution for single multi-processor machines is given in [Baum90], via a walk-through and a radiosity program. For the radiosity program, the usage of a hardware frame buffer is split, so that N equal size blocks (item buffers) are dynamically scheduled to perform item buffering. A ringed queue of data sets is shared between one producer and k consumer processes, such that each set has it own item buffers, and possibly form-factors. In their implementation a value of $k = 2$ is used, which is just double buffering. Each consumer computes form-factors from item buffer information and stores it in a local copy of a form-factor vector, for less access contention.

They note that only one producer is sufficient to drive the graphics hardware to capacity. Since the producer schedules the item buffers, and performs the projection of surfaces onto each buffer, most of the time is spent on rendering. This time is dependent on the number of hemicubes and polygons, the rendering rate, and the read back rate from the buffers, which in turn depends on the number of buffers, and the pixel resolution of the hemicubes. An interesting result is that it is possible to perform an *a priori* calculation of optimal values for parameters based on the number of polygons, hemicube resolution, and the number of consumer threads.

4.8.2 Form-factor Solutions

[Bu89] computes form-factors for planar diffuse polygons by ray tracing through the grid cells of a uniformly meshed hemicube. Every ray that strikes the interior of a polygon projected onto the hemicube is given a weight of 1, and every cell that strikes an edge of a projected polygon $\frac{1}{2}$. These weights are used to weigh associated delta form-factors at grid cells, in order to give better projected area approximations. Intersection computations have been pipelined, and rays to trace are computed in parallel.

Chapter 5

Radiosity Implementation

5.1 Introduction

In this chapter an implementation of a radiosity algorithm will be presented. The choices of techniques and ideas used are based on the analysis presented in the previous chapter, and implementation choices. First, a general overview of the implementation will be presented, followed by details for each section of the algorithm. A summary of the complete algorithm is given at the close of this chapter.

5.2 Overview

The basic aim of the algorithm is to provide an implementation which is useful on as many different hardware platforms as possible, with the emphasis on producing a simple usable application with sufficient flexibility to allow for further enhancement.

The basis of the algorithm has been taken from the following sources:

- Modeling uses ideas from [Cohen86] [Segal88] and [Baum91], and data produced by the **QuickModel** modeling program [Alias89].
- Visibility computation uses ideas from [Amana87], [Goldsmith87], [Haines91a], [Hanrahan91], [Arvo89].
- Form-factor computation uses ideas from [Wallace89] [Baum89] [Rush90b] and [Arvo91].
- The progressive refinement method with the hemicube, and ray casting techniques, and adaptive refinement makes use of [Goral84] [Cohen85] [Cohen86] [Chen89] [Arvo91].

- Rendering uses ideas from [Cohen86] [Wallace89] [Airey90] and [Sillion90].

The techniques are organized and incorporated in three basic parts:

1. A modeling part.
2. A radiosity simulation part.
3. A rendering and environment walk-through part.

The main reason for this particular breakdown is to treat the radiosity simulation as independent of the modeling and rendering parts of the process. This allows for extensions to accept various different types of data formats for models, and different renderers to be used for displaying scenes. The radiosity part may then compute simulations that are roughly independent of input model and output models, and any associated hardware required.

In general, modeling and rendering are treated as pre and post processing steps, in order to emphasize the radiosity simulation part. The simulation is hierarchically broken down into a top level progressive refinement component, with form-factor computation, adaptive subdivision, choice of shooting patch, and distribution of energy sub-components.

Figure 5.1 shows the basic breakdown of the entire system implemented.

As with the previous chapter on analysis, discussion sequentially follows the steps of modeling, radiosity solution computation, and then rendering.

5.3 Modeling

5.3.1 The Environment

The environments considered will be restricted to those that are best suited for view-independent walk-throughs. It is assumed that scenes are enclosures, (though not necessary), with ideal diffuse surface emittance and reflectance, and no participating media. Geared towards usage on machines that provide hardware-assisted polygonal rendering for speed, models have been restricted to being polygonal. In particular, polygons are planar, and are either quadrilateral or triangular in shape.

Given these assumptions, the central aim is to incorporate, in an automated fashion, some of the ideas presented in [Cohen86], [Segal88] and [Baum91] to produce models with various geometric, radiosity simulation, and rendering constraints best suited for these models. It is assumed that initial input models incorporate good modeling principles ([Mantyla88], [Samet90]).

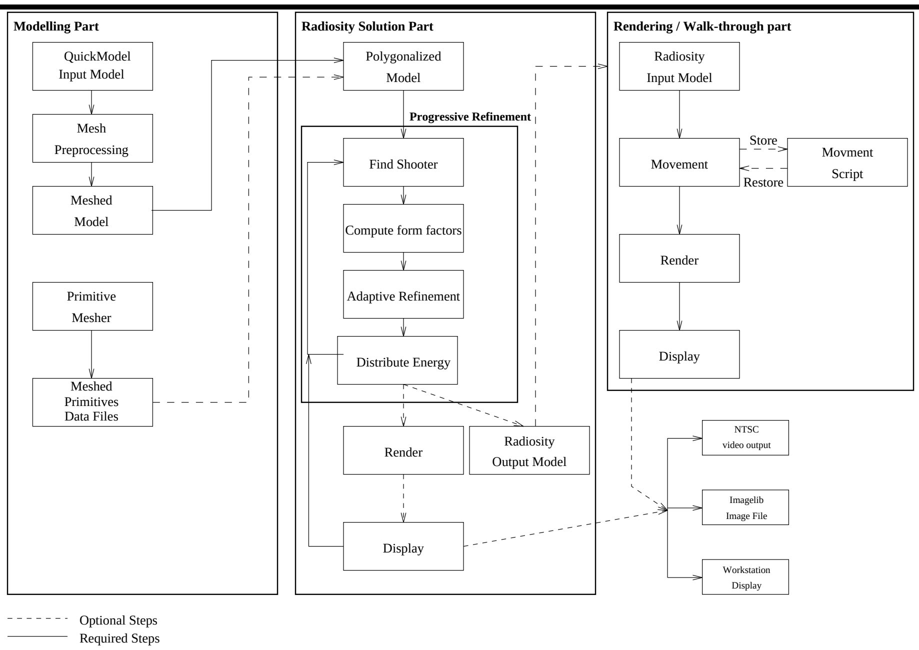

Figure 5.1: Breakdown of radiosity algorithm, mesh processors, and walk-through program.

5.3.2 Model Generation

An initial description of the scene is produced using a modeling program called **QuickModel**. The program is commonly available for most *Silicon Graphics IRIS* graphics workstations, and has a simple data format. The main features to take advantage of are:

1. A graphical interface for defining **objects** which are **non-primitive** meshes, and **primitives**. Primitives include *cones, cubes, cylinders*, and *spheres.*
2. Mesh information is represented as line segments. Non-primitive meshes represent *surfaces of revolution*, or *extrusions.*
3. Definition of RGB colour information, for each primitive.

Drawbacks of this modeler includes the fact that only point light source definition is included, which is unsuitable for our needs, surface properties are simplistic, and surface normal and polygon information is not implicit.

5.3.3 Model Enhancement

To handle some of these shortcomings, model preprocessing is performed. A preprocessor is used to create data files containing information for polygonally meshed spheres, cones, cylinders, and cubes. The degree to which they match the real primitives geometry depends on the degree of subdivision defined. Note that regardless of mesh resolution, upon display, objects may still appear polygonized depending on the position the mesh is viewed from. Currently, a default resolution that produces approximations that are fairly consistent with the actual geometry of the primitive is used. These data files are used for *primitive instancing*, in that each primitive has only one definition, and instances may be made of the primitive which contain references to these definitions. Additional preprocessing computes the following information from QuickModel data files:

1. In terms of geometric constraints, explicit polygons and unambiguous surface faces (which way is out), and normal consistency is computed. Normals for polygons point "out" if counting the vertices of that polygon in a counter-clockwise direction.
2. Well shaped polygons are formed by subdividing polygons either into rectangular or triangular shaped polygons. A user defined level of subdivision of patches to elements is allowed.
3. Interpretation of RGB colour information as ideal diffuse surface reflectance. Ideal diffuse emission information must be manually set, as there is no piece of information in QuickModel data files suitable to use.
4. Implicit access to the previously mentioned data files containing data for meshed spheres, cones, cylinders, and cubes.
5. Mesh vertex alignment between surfaces for primitives is included if they do not interpenetrate.

Restrictions in the model include the fact that only the "exterior" of faces radiate energy, and coplanar surfaces have not been removed except if they face each other, due to the lack of a good criterium for the general case. Mesh misalignment has not been handled by either balancing or anchouring ([Baum91]).

Following preprocessing, the basic output constructs are polygon normals, vertices, and radiative properties, with vertex and normal to polygon, polygon to polygonal mesh, mesh to primitive, and primitive to environment relationships.

5.3.4 Model Storage and Filters

To use models stored in different data formats, filters may be used to translate to the appropriate input format. An example filter is currently provided for models stored in **OFF format** ([Rost86]).

5.4 Radiosity Structures

The output produced by the modeling process represents the input models for our radiosity program, where additional preprocessing is performed to create a structure hierarchy, vertex tree and space subdivision as shown in *Figure 5.2*.

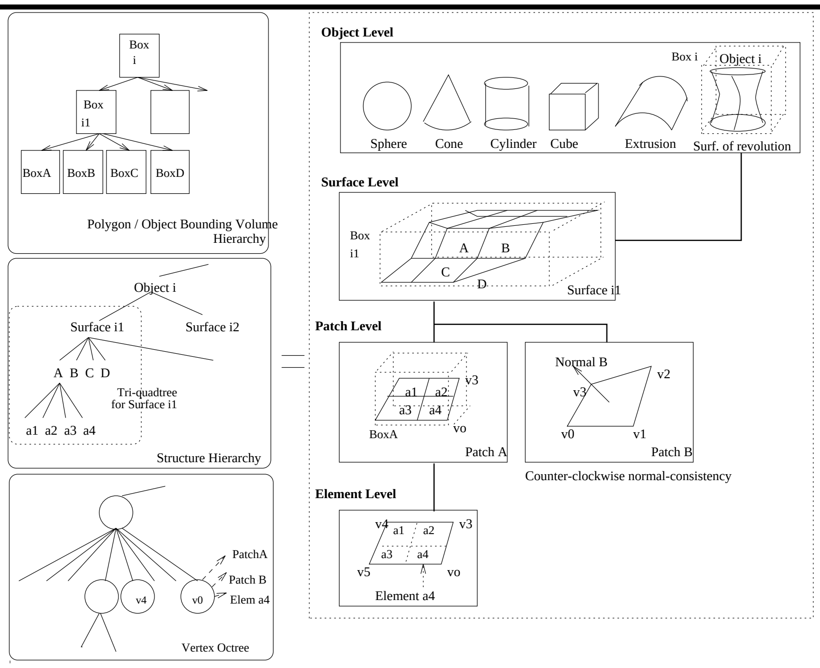

Figure 5.2: Breakdown and relationships for vertex octree, hierarchical bounding-volume trees, and structure hierarchy

5.4.1 Structure Hierarchy

From the input models, a hierarchy of the form shown in *Figure 5.2* is created. This hierarchy is maintained: a) for the sharing of information between levels, reduced storage costs, and maintaining consistency in the model; b) for adjacency information and geometric relationships; and c) to allow working at the desired level of detail for different algorithms, in particular for form-factor calculations, and selection of source and receivers. Relationships maintained between entities include:

1. Mesh to algebraic definition: Algebraic representations are kept for primitives (cones, spheres, cylinders and cubes).
2. Patch to element: A *tri-quadtree* is used to capture surface subdivision, allowing sharing of surface information and adaptive subdivision.
3. Patch to mesh to object: for sharing of surface property information, and geometric relationships.
4. Vertex to patch or element: To maintain adjacency information a *vertex tree* ([Baum91]) is used. This is an octree of unique vertices, where each non-empty octant contains a vertex definition. Uniqueness is determined by vertex position within a given tolerance volume.

Model-to-scene transformations, algebraic definition usage, and vertex tree usage are discussed in later sections.

5.4.2 Object Coherence

To exploit spatial properties between objects in the scene, a hierarchical bounding-volume tree ([Weghorst84]) is used. This method has been mainly chosen for tightness of volumes with respect to objects.

A *bounding volume (BV)* is a simple surface that is used to enclose one or more objects or bounding volumes. Axis-aligned *bounding boxes* have been chosen for their simplicity, with respect to culling, and ray-intersection testing. A *hierarchical bounding-volume tree (HBV tree)* may be defined as containing nodes, such that a leaf node is either empty or an object and all other nodes are bounding-volume trees.

A tree is automatically created, using the constraints in [Goldsmith87], by adding bounding volumes for each object one at a time such that the location in the tree to insert the volume at causes the minimal increase in "cost" to all ancestors of the newly created node. As noted by Goldsmith, the area increase of a volume may be used as a measure of cost, as long as all volumes have the same geometry. A two level hierarchy is actually created, the first being a tree of bounding volumes for objects (*object HBV*), and the second being sub-trees containing bounding volumes for polygons of each **non-primitive** object (*polygon HBV*). It is assumed that adding polygons in a hierarchy below an object node will have the least incremental cost. These two levels are used so that work may be done at the desired level of detail. The reason as to why only meshes for non-primitives are included will be given when discussing ray-intersection testing.

5.4.3 Surface Properties

The surface properties implemented cannot capture a number of different effects, but are simple and provide for fast computation. Ideally, properties should be independently defined in a continuous representation which requires low storage, and that can easily be mapped to calculation points. For the most physical accuracy, Sillion's representation ([Sillion91]) using illumination models such as [He91] [Cook82] or [Hall83] is suggested. Though less accurate, less expensive ideas such as extensions of bump maps or texture mapping ([ChenH90] [Cohen86]) may be used.

5.5 Progressive Refinement

In order to solve for diffuse global illumination, a system of equations which gives the dependence of energy leaving each surface on the impinging energy from every other surface needs to be solved. Assuming ideal diffuse properties for all surfaces, the energy leaving a surface i is given by:

$$B_i A_i = E_i A_i + \rho_i \sum_{j=1}^{n} B_j A_j F_{i-j} \tag{5.1}$$

where

B_i, B_j	=	radiosities at surfaces i and j (energy per unit area)
A_i, A_j	=	areas of surface i and j
E_i	=	emitted energy per unit area
ρ_i	=	reflectivity of surface i
F_{i-j}	=	form-factor from surface j to surface i

In this formulation the energy at surface i is dependent upon all other surfaces j sending energy to i before the energy at the surface may be updated.

In our implementation a progressive refinement approach has been incorporated ([Chen89]). In this method, the reciprocity rule for form-factors:

$$F_{j-i} = F_{i-j} \frac{A_i}{A_j} \tag{5.2}$$

is used to compute the energy contribution at all other surfaces j due to energy leaving surface i.

For all surfaces j the following is computed:

$$B_j = B_j + B_i\,(\rho_j F_{j-i}) \tag{5.3}$$

where F_{j-i} is computed from equation (5.2).

An **iteration** of progressive refinement consists of selecting a *source* patch or *shooter* (i), and then computing B_j for all *receiver* patches (j). From [Cohen88] the algorithm presented which allows for adaptive subdivision of receivers, has been adopted. The basic difference is that energy is propagated from a source patch to receiver <u>elements</u> as opposed to receiver <u>patches</u>. The radiosity for receiver patches is found by taking area-weighted averages of it's element radiosities.

As noted by Cohen, only form-factor storage linear to the number of elements is required per iteration, and that adaptive refinement of images at every iteration is achieved.

5.5.1 Choosing the Shooting Patch

The shooting patch for a given iteration is the patch with the current maximum unshot energy. If the environment is dynamic, then the absolute unshot energy may be used instead, with the additionally criteria of choosing the least recently moved patch, or the most recently shadowed, if the viewer is stationary ([Malley88], [Sillion89], [Chen90]). Otherwise, patches that effect what viewer will see should be given higher priority ([Baum90]). Selection is only by maximum unshot energy, as static environments are assumed.

To handle problems such as the fact that few or no objects may be visible to a shooter ([Haines91a]) the definition of a receiver must be refined, rather than naively assuming that all patches that are not the shooter are receivers.

5.5.2 Choosing Receiver Patches

Once a shooter has been chosen, a list of receivers is required. This list is mainly kept to avoid recomputing the possible receivers that are visible to the current source. That is, the identification of receivers does not need to be recomputed for each ray-visibility test, and each form factor computation performed for the current shooter. In order to refine this list, both spatial and surface properties, have been exploited.

When attempting to determine visibility between two surfaces, visibility is only considered from the front face of each surface. Since planar patches are assumed, everything that is behind the plane the patch lies on is not visible, and therefore need not be considered (or is *back-face culled*). Using this idea, the object hierarchical bounding-volume tree is used to first determine a set of objects that may be visible to the shooter:

Step A. If the node is a bounding volume, then back-face culling is performed on this volume and possible sub-volumes recursively, until a volume is found that is either partially or fully in front of the shooter. Step B is then performed on the object for any volume found.

Step B. For tree nodes that are objects: If the object is a primitive and is behind the source patch, then all patches on that object may be discounted, since all primitives are convex and the shooter cannot be visible to any patch on that object. Otherwise, the object is added to a list of possible object receivers we will call the **object receiver list** or **OR-list**.

As a second step, the OR-list is traversed to exclude patches or objects as follows:

1. Exclude objects that will not reflect any of the radiosity from the source, in all wavelength bands. A better method tolerance, may be to exclude those with an (energy $\times$ reflectance) value smaller than some minimal amount.
2. Back-face culling is used to remove any polygons that do not face the shooter, or are behind the shooter.
3. Discount the shooter, and elements of the shooter.

If a patch is composed of elements, only elements of the patch that pass all constraints are included. All receivers are kept in a receiver list **(R-list)**.

At present, specular surfaces are not included. For specular surfaces, the above culling criteria may still be applied iteratively for every bounce off of a specular surface, wherein culling is performed with respect to each specular surface encountered. A more complex, or different scheme would be required if objects are not opaque. An object is *opaque* if all impinging energy is either absorbed or reflected by the surfaces of that object.

5.5.3 Convergence Criteria

The current implementation uses the threshold given in [Chen89], of halting when a shooting patch that has more than $\frac{1}{1000}$ th of the original energy left to propagate cannot be found. This tolerance may be redefined by the user. The simulation may also be halted after a specified number of iterations. By default, a maximum value of 100 iterations is assumed, according to experimental results ([Chen89], [Cohen88], [Greenberg91]) though this value is highly dependent on the geometry and surface properties in the scene.

5.6 Form-factors

Three different form-factor formulations from [Wallace89], [Baum89] and [Sarofim67], and [Siegel81] will be examined with respect to visibility determination and form-factor computation. Adaptive subdivision and progressive refinement are accommodated, by sampling from element vertices to source patches to compute form-factors. These techniques are applicable for opaque surfaces.

5.6.1 Formulations

Disc Formulation

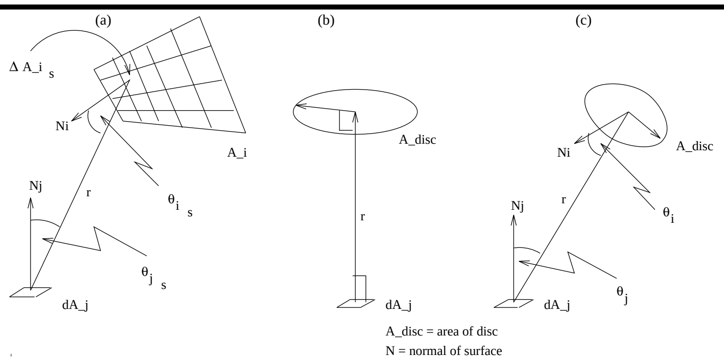

Figure 5.3: a) Numerical integration of form-factor from differential area j to finite area i. Sampling of source A_i from $\triangle A_{i_s}$ sample areas using form-factor geometry shown in c). b) Differential area to parallel disc geometry. c) Differential area to arbitrarily oriented disc geometry.

Wallace's form-factor approximation ([Wallace89]) is chosen for its appropriateness for polygonal or non-polygonalized models, ability to sample using real surface geometry, adequate sampling of sources, adaptive sampling, and simple formulation. In this method a source (i) is subdivided into a number of sample areas ($\triangle A_{i_s}$), with a sample point located within each sample area. For every vertex of every receiver (j), the form-factor for each sample s is given as:

$$F_{\triangle A_{i_s} - dA_j} = dA_j VIS \left(\frac{cos\theta_{j_s} cos\theta_{i_s}}{\pi r_s^2 + \triangle A_{i_s}} \right) \tag{5.4}$$

where

dA_j	=	differential area at the vertex of the receiver
$\triangle A_{i_s}$	=	area weight of the sample i
θ_{j_s}	=	angle between normal at dA_j and direction to $\triangle A_{i_s}$
θ_{i_s}	=	angle between normal at $\triangle A_{i_s}$ and direction to dA_j
r	=	the distance between the areas
VIS	=	1 if the sample point is visible to the vertex, 0 if occluded

To find the form-factor from the whole source to each vertex an area weighted average may be performed.

$$F_{A_i - dA_j} = dA_j \frac{1}{n} \sum_{s=1}^{n} VIS \left(\frac{cos\theta_{j_s} cos\theta_{i_s}}{\pi r_s^2 + \triangle A_{i_s}} \right) \tag{5.5}$$

where n is the number of source samples.

From equation (5.1), the radiosity at surface j due to energy from surface i is given as

$$B_j A_j = \rho_j B_i A_i F_{i-j} \tag{5.6}$$

Substituting equation (5.5), the radiosity at vertex j due to energy from source i is given as:

$$B_j = \rho_j B_i \left[\frac{1}{n} \sum_{s=1}^{n} VIS \left(\frac{cos\theta_{j_s} cos\theta_{i_s}}{\pi r_s^2 + \triangle A_{i_s}} \right) \triangle A_{i_s} \right] \tag{5.7}$$

There are some problems inherent in this approximation. As Wallace notes, equation (5.4) extends a true analytic formulation for a differential area to finite parallel disc by introducing cosines of angles between the normal at each surface, and the direction between the source and the receiver (*See Figure 5.3*). Thus errors occur due to differences between the true shape of the source and a disc (e.g. if the source is elongated) , and non-parallel orientations. These errors are noticeable when sampling at rates lower that 16 source samples per vertex ([Wallace89]).

An example of orientation producing sampling problems is seen in *Figure 5.4* where the source and receiver are at right angles to one another and abut. An illumination gradient in the opposite direction occurs. This is due to the fact that sampling is only performed from vertex positions, and at vertices where the patches touch no energy is transferred. Thus, more energy appears to be transferred to areas further away from the source than those closer, since interpolation between samples is used. A potentially large amount of sampling may be required to capture the correct gradient in such situations. Note that no matter how much adaptive subdivision is performed, a reverse illumination gradient will still occur.

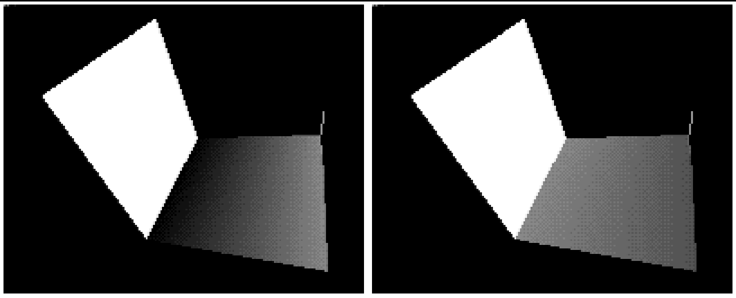

Figure 5.4: Figure on left shows reverse gradient on receiver polygon sampling perpendicular source using Wallaces approximation. Figure on right shows use of Baums analytic approximation on same geometry.

Analytic Formulation

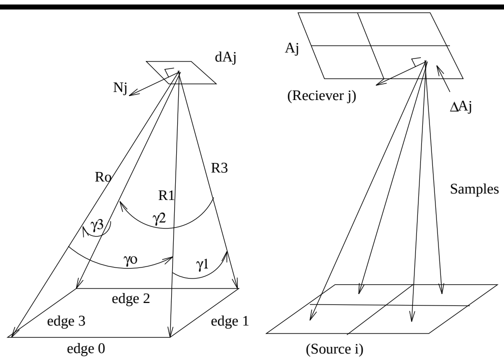

Figure 5.5: Left shows geometry for evaluating analytic form-factor. Right shows sampling geometry from a receiver to source to compute visibility for form-factors. (After [Baum89])

To handle the above sampling problem, an analytic form-factor approximation is used ([Baum89]), which uses the differential area to unoccluded finite area polygon approximation found in [Sarofim67]:

$$F_{dA_j - A_i} = \frac{1}{2\pi} \sum_{g \in Gi} N_j \cdot \Gamma_g \tag{5.8}$$

G_i is the set of edges of surface i, N_j is the surface normal for differential surface j, Γ_g is a vector with magnitude equal to the angle γ_g (in radians), and direction given by $R_g \otimes R_{g+1}$, (where $\otimes$ is the vector cross product), as shown in *Figure 5.5*.

To account for occluding objects, Baum uses a hemicube to compute visibility. For every receiver element, the pixel coverage is used to determine sample positions on the element, and an area weight for the analytic form factor computed from each sample position. To incorporate this method with ray casting, equation (5.8) is computed from uniform sample points on receiver elements, with visibility computed by sampling from receiver sample points to uniformly distributed sample points on the source. The percentage of samples seen by the source is used as the percent visibility, and the receiver sample area is used to weight each form-factor. As with Wallace's formulation ([Wallace89]), a similar equation can be derived to compute the radiosity at vertex j due to energy from source i:

$$B_j = \rho_j B_i F_{\triangle e_j - Ai} \tag{5.9}$$

where

$$F_{\triangle e_j - Ai} = F_{dA_j - A_i} \left(\sum_{s=1}^{S} VIS \right) \left(\frac{\triangle A_{e_j}}{A_j} \right) \tag{5.10}$$

S is the number of samples to compute visibility, and $\triangle A_{e_j}$ is the receiver sample area.

For both Wallace's and Baum's form-factor formulations, using averaged area weighted samples avoids the sampling errors that result from counting samples to compute visibility ([Chen89]), and the proximity assumption ([Baum89]) is maintained since form-factors are computed from differential areas.

Elliptical Formulation

To handle problems with elongated surfaces, an extension of a differential area to parallel ellipse formulation ([Siegel81]) has been proposed. Using a similar derivation as in [Wallace89], for every vertex of a receiver (j), the form-factor with source sample i is given as:

$$F_{\triangle A_{i_s} - dA_j} = dA_j VIS \left(\frac{cos\theta_{j_s} cos\theta_{i_s}}{\pi(a_s^2 + r_s^2)(b_s^2 + r_s^2)} \right) \tag{5.11}$$

where a and b are the approximate major and minor axis of an ellipse that would fit the source sample's (i) area. This method, has a trade off between more expensive area computation, versus more samples per form-factor.

Hemicube Formulation

For fast initial approximate solutions, a hemicube form-factor formulation ([Cohen85]) is included, which uses the graphics hardware found on *Silicon Graphics Iris* workstations. For accurate results, this method is not supported, due to problems previously mentioned in the analysis. This formulation basically approximates the form-factor from patch i to patch j as (from *Figure 2.3*) :

$$F_{i-j} \cong \int_{A_j} \frac{cos\theta_i \cos\theta_j VISdA_j}{\pi r_{ij}^2} \tag{5.12}$$

which assumes that the distance between the two surfaces is great compared to the area of the receiver, and that the form-factor from the center of the source applies to the whole surface. Using a hemicube, F_{i-j} is discretely computed as:

$$F_{i-j} \cong \sum_{q \in Q_{ij}} \triangle F_q \tag{5.13}$$

where Q_{ij} is the set of hemicube pixels through which surface j is visible to the center of the surface i, and $\triangle F_q$ is the *delta form-factor* associated with pixel q. Aliasing problems for the hemicube have been addressed in part by using a *jittered hemicube* ([Arvo91]), though aliasing still results from uniform sampling (See *Figure 5.6*). We refer you to the respective references for further details.

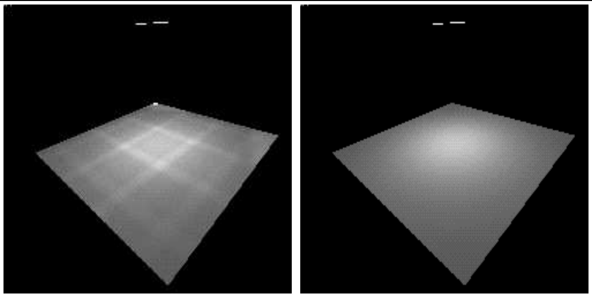

Figure 5.6: On the left, aliasing due to uniform sampling imposed by the jittered hemicube, is shown. The surface is subdivided into 32 by 32 elements. On the right the sample geometry with form-factors computed using Wallaces method (After [Wallace89]).

5.6.2 Sampling and Visibility determination

A crucial, and time consuming step is to compute the visibility between surfaces. Currently a point sampling strategy is used to determine visibility between two points. Surface to surface visibility is determined by counting samples. As in ([Wallace89]), the source is bilinearly subdivided into a number of sample areas, whose centers are used as sample positions. Noting that source and receiver subdivision may not be compatible, sources are not actually subdivided permanently, but only to derive sample points, and area weights for form-factor computation.

Sampling strategies using either *uniform* or *jittered* sampling positions has been implemented, but adaptive source sampling has not been implemented. Adaptively adjusting the source sample density will be covered when discussing capturing gradient information.

Neither form-factor nor visibility computation information is currently reused, resulting in duplicate computation. A proposed scheme is to store form-factors or visibility at the vertices for reuse on a per visibility determination basis as in [Sillion89]. Note that polygons that share a vertex must have the same orientation in order to share form-factor information at that vertex. In addition, ideas from [Hanrahan91] to reduce the number of form-factors to compute and store has not been incorporated. This should greatly reduce computation and storage requirements.

To compute visibility, ray casting is used, where each sample corresponds to a ray cast between a receiver element point and a source sample point. The percentage visibility is the fraction of rays not intersected by some object before reaching the source sample positions, divided by the total number of rays shot (total samples taken).

5.6.3 Accelerating Ray Casting

As with choosing receivers, all objects in the scene should not be naively tested against every ray used for sampling. By making use of the bounding volume hierarchy, as well as exploiting aspects of the structure hierarchy, the following ideas are implemented:

- Using higher order representations (algebraic surfaces) to reduce the number of intersections.
- Exploiting object coherence, through the bounding volume hierarchy.
- Exploiting constraints in space when computing visibility via back-face and shaft culling.

Ray-Primitive Intersection Testing

To reduce the number of intersection tests required, ray intersection tests are performed using the algebraic definitions of primitives, rather than against every polygon in the mesh representing the primitive. The true surface geometry of the surface is used rather than a polygonal approximation.

To simplify the intersection computation, tests are performed in the coordinate system of the primitive instead of the coordinate system of the scene, by transforming the sample ray using a model-to-scene transformation matrix kept with every primitive. Since all primitives are instances of representations that fit within a unit cube centered around the origin, the ray-primitive intersection computation is simplified. A fixed amount of time is required for each intersection test, since transformations that affect orientation and size of the primitive are not present. The above ideas and formulations for ray-primitive intersections are taken from the **Optik** ray tracing program ([Amana87]).

Object Coherence

A common technique to reduce the number of ray-object intersection tests is to exploit object coherence. To avoid testing potentially complex objects, the associated geometrically simpler bounding volume may first be tested. If the object is already simple, as with the primitives implemented, there is no need to test against the primitive's volume first. For non-primitives, where arbitrarily shaped polygonal meshes are present, the cost of testing against a bounding volume first is justified. Using similar reasoning as for the usage of algebraic representations, time may be saved by testing against a single volume first, versus all of the polygons of a mesh. The cost per volume test depends on the the complexity of volume, and how tightly the volume fits the object. Though boxes have been chosen, more complex and tighter fitting volumes may be used ([Arvo89]).

To potentially further reduce the number of intersections a hierarchy of higher level groupings of volumes in space is used. That is, it is possible to disregard *groups* of objects if the ray does not intersect the volume. When considering the cost of creating and traversing this tree, an additional factor to consider is that this hierarchy is not solely used from ray-intersection testing. The $O(nlog(n))$ cost of building the tree is not considered as part of the ray-intersection test cost, but as part of the preprocess cost ([Goldsmith87]). The savings is that in the best case, only $O(log(n))$ intersection tests are required per ray to find an intersecting object.

Shaft Culling

Before performing any intersection tests, Haines ([Haines92a]) notes that the number of volumes to check per form-factor computation may be reduced by exploiting the fact that the visibility to determine is restricted to a finite volume between two surfaces.

As with receiver patch selection, back-face culling may be used to cull away non-visible parts of the scene. On a hierarchical basis, culling is performed in a top-down traversal of the object HBV tree. The aim is to form a list of "candidate" bounding volumes, which may be intersected by *any* ray used for sampling the source. This list will be called the *candidate list*. The bounding volume to test is called the *test bounding volume* or TBV, and the source and receiver bounding volumes are labeled SBV and RBV respectively.

The first test performed is to back-face cull the TBV against the source, and the receiver planes. If the volume survives this test it is then tested against the source, and receiver volumes, and a construct known as an *extent box* which is the minimum bounding box of the source and receiver boxes. The TBV is checked for overlap or inclusion in the extent box. If this is true, then if the TBV overlaps or contains the SBV or RBV then it is a potential intersecting volume and no further tests are performed, otherwise a structure called a **shaft** is used. The shaft is basically the convex hull of the bounding volumes of source and receiver surface. Since bounding boxes are assumed, it is quite straightforward to build the shaft as a series of connecting planes. The planes are used to back-face cull the hierarchy under the bounding volume, keeping those volumes that are either inside or overlapping the shaft. Various strategies are used as to how to perform testing on the hierarchy:

- Always open: always check all boxes in the hierarchy against the shaft.
- Overlap open: if the TBV overlaps the shaft, then check at the next level down.
- Keep closed: if the TBV overlaps the shaft, then don't check further down
- Ratio open: if the TBV overlaps by a certain percentage = percentage of children that overlap or are inside the shaft, then check the next level down.

Haines reports the best experimental results with a ratio open strategy of "opening" on 40 percent.

Primitives that are candidates are immediately added to the candidate list if they either overlap or are inside the shaft, the SBV or the RBV. If the primitive candidate contains the shaft, then it may be discarded if it is a cube, since nothing inside the cube can intersect with the shaft, otherwise the primitive is added to the list.

To handle non-primitives that contain the extent box, the tree is traversed until a bounding volume is found that does not contain the extent box, throwing away all volumes along the way. If that bounding volume cannot be found, then the original volume is added to the list.

As noted by Haines, the complexity of this testing may be justified since shaft culling need only be performed once per source-to-receiver visibility test regardless of the ray/sample density. Currently a minimum of three rays is tested per shaft for a triangular receiver. Also, the shaft should in practice be thin, since shafts are formed between elements and patches.

5.6.4 Ray-Intersection Testing

Using the above acceleration techniques the algorithm to check for intersections with a ray for visibility is given below. The maximum distance that the ray may travel (from the receiver position to the source sample position) is labeled R_{max}. If a ray intersects any object or patch at a closer distance, than searching halts.

Intersection Algorithm {
if using bounding volumes then {
  if using a shaft then for all candidates in candidate list {
    If the candidate is a polygon bounding volume then
      Check the polygon HBV tree starting at the candidate bounding volume
    else
      Check the object HBV tree starting at the candidate bounding volume
  } else if back-face culling using source and receiver {
    Check against volumes in front of receiver and source in the HBV tree
  } else
    Check against all volumes in the whole HBV tree starting at the root.
} else
  Perform ray-object intersection tests for all objects. Stop if an intersection closer than R_{max} is found
}

Object Bounding Volume Testing {
While the ray has not intersected an object at distance $< R_{max}$ {
If testing against an object then
    Perform ray-object intersection testing
  else
    Check for intersection with bounding volume
    If intersection distance $< R_{max}$ then
      Perform bounding volume testing on any children
  }
}

Ray-Object Intersect Testing {
If object is a primitive then
  Do intersection test in primitive coordinate system

```
else if object is a mesh then
  if using bounding volumes then check polygon HBV
  else
    Check against BV of polygon before checking against
    the polygon itself.
Stop if intersection distance < R_max
}
```

For every ray, the main intersection algorithm is performed to determine visibility between the origin, and destination of that ray. Note that the testing for the polygon HBV tree is similar to that for the object HBV, except that polygons and their associated BVs are tested versus objects and object BVs. Also there is no primitive testing.

5.7 Capturing Gradients

5.7.1 Adaptive Subdivision

In order to capture radiosity gradients across surfaces, an adaptive subdivision method ([Cohen86]) is used. A two-fold criteria of form-factor and radiosity gradients is used to determine when to subdivide. If the difference between the form-factor or radiosity values of two vertices sharing an edge is greater than a specified tolerance, then subdivision may be performed. The form-factor test is used to capture visibility gradients that may be missed by simply using a radiosity gradient criteria ([Arvo91]).

To maintain well-shaped polygons, bilinear subdivision is used for both triangular and quadrilateral patches or elements, with monotonic change along the edge being assumed when assigning values to new subdivision points. As previously mentioned, neither tree balancing nor anchouring has been performed to fix mesh problems ([Baum91]).

To avoid excessive meshing of receivers, a minimum receiver area is used, which is specified as a percentage of the average area of a polygon in a scene. The average area is computed as the ratio of the total area of the scene over the number of polygons. Patch subdivision is maintained in an n level tri-quadtree ([Baum91]), with the area of elements decreasing bilinearly for each level down the tree. A second criteria to avoid excessive receiver meshing, is a user-defined maximum subdivision level.

To handle problems with using the reciprocity rule in the hemicube form-factor implementation, the criteria from [Chen89] of checking for form-factors greater than one is used. Currently no adaptive subdivision is performed on walk-throughs (e.g. [Sillion89]), since the walk-through program is disjoint from the simulation implementation.

5.7.2 Adaptive Sampling

To take into account the amount of subdivision performed on receivers, the number of source samples per vertex is adjusted to be inversely proportional to the level of receiver subdivision. This is to maintain roughly the same amount of sampling per unit area of a receiver patch, and to avoid undue cost for squared increases in sampling rate due to bilinear subdivision.

The number of samples with respect to the source may also be adjusted. To avoid undersampling the source, the number of samples taken may be adjusted based on the radiosity received at the vertex of the receiver, and resampling at a higher rate if the radiosity is too large ([Wallace89]). As Campbell notes ([Campbell90]), oversampling should be avoided as well. A combined approximate method is currently used, using the solid angle that the source subtends with respect to the receiver ([Campbell90]), and the amount of radiosity the source may send to the receiver with respect to the total energy left. If the solid angle is too small, then the number of samples is reduced; if the radiosity is too large, then the number of samples is increased. Currently a minimum threshold of 0.005 steradians is used as in ([Campbell90]). Any light subtending a smaller angle is considered to be a point light source.

For more efficient sampling, as previously mentioned, samples for visibility and form-factor calculations should be reused.

5.7.3 Non-diffuse Environments

The above sampling methods have been used to capture radiosity gradients at patches with ideal diffuse surface properties. For more complex reflectance or emittance properties for the receiver or source, consideration should be given to weighting the distribution of samples by these properties to avoid oversampling in directions where little energy may be exchanged. Currently, probabilistic sampling is suggested as the most suitable approach to capture general surface properties ([Greenberg91]), using some form of distributed ray tracing.

5.8 Rendering

5.8.1 Reconstructing Radiosity Gradients

Since two methods are used for computing form-factors, values are stored differently for each method.

If a hemicube method is used, radiosity values are stored per patch or element. We will call this *patch storage*. Otherwise, radiosity values are stored per vertex, such that if a vertex is shared by more than one element or patch, with different surface properties or different orientation, then more than one radiosity value is stored. We will call this *vertex storage*.

Using patch storage, radiosity values at vertices are computed by bilinearly interpolating the values for patches or elements sharing the vertex, that have the same surface properties, and do not contain elements that also belong to this vertex. That is, the smallest shared polygons are always used to interpolate. If vertex storage is used, an area-weighted average of all values stored at that vertex is computed, again, for the smallest shared polygons with the same surface properties.

In general, better reconstruction methods may be employed such as higher order interpolation, or filtering to produce more accurate radiosity gradients (e.g. [Haines91a]). In addition, radiosity values are currently stored at fixed points (vertices). For a freer choice of points, texture storage could be used ([Heckbert90]), with separation of rendering and radiosity storage structures. This allows true surface representations to be used for rendering versus polygonal breakdowns.

5.8.2 Shading

Given a polygonized model with values only at vertex points, polygons are currently displayed using Gouraud shading ([Gour71]). This method is used due to the fact that input models contain surfaces with ideal diffuse surface properties, and hardware implementation of this method, allowing for sufficient speed for real-time walk-throughs.

Shading problems associated with vertex misalignment in polygonal meshes have not been handled. To handle problems with rendering quadrilaterals, which are rotation variant, a post process of triangulating all quadrilaterals as close as possible to the intensity gradient has been used. As in [Airey90], each quadrilateral is split by connecting the two opposing vertices with the least radiosity difference. If there is a tie, an arbitrarily choice is made.

If non-ideal diffuse surfaces are considered, an alternative method would be to use ray tracing, with interpolation or filtering of radiosity values at surface intersection points (e.g. [Shirley90], [Chen91]).

Spectral Sampling

Three wavelength bands are used to compute three sets of radiosity values, which are interpreted as the red, green and blue channels of a RGB colour space. It is possible to use more wavelength bands for better more accuracy (four is suggested in [Hall89]), though more time is required to compute the distribution of radiosity. Currently, user-defined gamma correction is included to scale the colour of images produced for different hardware displays.

Ambient Term

An ambient term, from [Chen89], which uses a form-factor and reflectance approximation is computed. Display values are a combination of the radiosity per patch or vertex (i) and the ambient term as follows:

$$B_i^{display} = B_i + \rho_i Ambient \tag{5.14}$$

The term is **not** used to compute the radiosity solution, and is scalable so as not to wipe out any subtle radiosity gradients, in rendered images.

5.8.3 Iterative Image Refinement

To view progression of the simulation, intermediate solutions are displayed after every progressive refinement iteration. Display is currently restricted to Silicon Graphics Iris workstations, that have sufficient hardware to display RGB colour information. The scene may be viewed from a user-defined viewing position, orientation, and field of view.

5.9 Radiosity Algorithm

The complete algorithm is summarized in the following pseudo-code description. Delimiters () denote optional steps that are user-defined, or restricted by hardware platform.

```
/* Initialization step: */
Read and preprocess initial model;
Read user options;
if using bounding volumes then
  Create object and polygon bounding-volume trees
```

```
(Initialize ray tracing, shaft culling, and hemicube statistics);
(Calculate "ambient term");
(If displaying on screen)
  (If using hemicube, create hardware item-buffer);
  (Display initial image of model);

/* Iterative radiosity solution step: */
Sort and find shooting patch (S);
While (radiosity stopping criteria is not met) {
  if using bounding volumes
    Compute object receiver list (OR_list) and receiver list (R_list) from OR_list;
  else
    Compute receiver list (R_list) from all objects;

  /* form-factor computation step: */
  For all receivers (R ∈ R_list) {
    if using the hemicube method
      Compute form-factor F_R-S from receiver to source using item-buffer;
    else
      If using shaft, create shaft and perform shaft culling;
      Decide on which form-factor (disc or analytic) to compute;
      Adjust sampling rate for visibility testing;
      Compute form-factor F_R-S from receiver to source using ray casting;
    Subdivide R if the form-factor gradient is exceeded or (F_R-S > 1);
    Calculate and propagate energy from S to R;
    Subdivide R if radiosity gradient exceeded;
  }

  (Calculate "ambient term");
  (Compute vertex radiosities and display intermediate images);
  Sort and find shooting patch (S);
}

/* End process */
(Display final image);
(Output model with final radiosity values);
```

5.10 Walk-Throughs

A simple walk-through program has been provided to view the scenes produced by the radiosity program. Polygons containing colour information per polygon or for each vertex of a polygon may be read in. Options provided include rotation around the axises in the coordinate system of the viewer, and translation along or perpendicular to the the viewing direction for a "zooming" effect. Viewing parameters and the movements listed above may be performed interactively or stored and retrieved for automated movement through the scene. Viewing parameters include position, orientation, and field of view.

Rendering options include using either Gouraud, flat shading, or viewing the polygon mesh, an option to triangulate quadrilaterals into triangles, and storage and retrieval of images in *IRIS imagelib format*.

Chapter 6

Experimental Results

6.1 Setup

The radiosity algorithm introduced is implemented in C on a Sun Sparcstation 2 workstation with 64 megabytes of main memory running the $Unix^{TM}$ operating system. and a Silicon Graphics Iris 4D workstation with 16 megabytes of main memory running the $Irix^{TM}$ operating system. The radiosity simulation code is 12364 lines long, the mesh preprocessors 2242 lines, and the walk-through program 3995 lines long. A optimized stripped version of the radiosity program, compiled on the Sparcstation, takes up 119907 bytes of disk storage. The amount of memory required by the simulation as a function of the number of polygons is roughly 0.6 kilobytes per polygon. All projection/visibility/depth-buffer computations for the hemicube method and rendering are performed using the $GeometryPipeline^{TM}$ and hardware Z-buffer ([Haeberli90]). All other computations are not dependent on any graphics hardware.

A number of different environments are chosen that range in terms of the number and size of polygons in the scene, the types of objects in the scene, and the kind of coherence that can be exploited in the scene. Results are presented for a cubic mostly empty enclosure, a mostly filled enclosure with several primitives, and two more complex scenes with roughly an even mixture of primitives and non-primitives. The first has a small ratio of emitters to non-emitters, and the second the reverse.

Statistics have been collected under a number of different categories:

1. Convergence rates: Graphs of time (number of iterations) versus total unshot energy left in the scene. Initial unshot is just the sum of all emission in the scene.
2. Ray casting information: Included is the total number of rays shot; the average number of ray tests and intersections per ray, primitive, polygon, or bounding volume; and the average number of intersections with the same entities per ray test.
3. Execution time: CPU usage of the total progressive refinement time, hierarchical bounding volume preprocessing time, average time per iteration, per shaft cull, per form-factor and per ray visibility test is recorded. All computation times are measured (in seconds) on the Sparcstation.

Comparisons are made using a set of default options: Use of object and polygon hierarchical bounding volumes is assumed. Wallace's numeric ([Wallace89]) and Baum's analytic form-factor ([Baum89]) are used, along with adaptive sampling, source and receiver back-face culling and shaft culling with a 40% ratio-open strategy. A minimum difference between form-factors of 0.015, and minimum polygon subdivision area of 10 percent of average polygon area is allowed. Computation of the ambient term and rendering are not included as part of the computation time. The stopping threshold is 1/1000 th of the original energy, or a maximum of 200 iterations.

A set of tests with the following variable options were performed:

1. Using 16 samples per vertex, the result is labeled **WALK** in tables of statistics given in *Appendix A*. These results are used to measure the convergence rates, and produce output images. Solutions using 16 samples, but without shaft culling, were also performed.
2. Testing using four shaft strategies: "Keep closed", "Always open", "Overlap open" and 40% "Ratio open" was performed. These tests are labeled **M0R4S0A1, M0R4S1A1, M0R4S2A1**, and **M0R4S3A1** respectively in *Appendix A*.
3. Testing at 1, 4 and 16 samples per vertex was performed, These tests are labeled **M0R1S0A1, M0R4S0A1,** and **WALK** respectively in *Appendix A*.
4. Testing without an adjusted sampling rate, for adaptive subdivision to zero, one or two levels was performed. These tests are labeled **M0R4S0A0, M1R4S0A0**, and **M2R4S0A0** in *Appendix A*. Two level subdivision is only performed for the simple cubic environment.
5. Testing with a maximum of 1 or 2 subdivision levels, and minimum area of 50 or 10 percent of the average polygon area for a given scene was performed. Corresponding tests without adaptive sampling for the simple cubic environment at 1 subdivision level are labeled: **M1R4S3A0a5** and **M1R4S3A0**, and at 2 subdivision levels: **M2R4S3A0a5** and **M2R4S3A0**. For all other environments, testing at 1 subdivision level was performed with tests labeled: **M1R4S3A1a5** and **M1R4S3A1** in *Appendix A*. **a5** denotes the 50 percent case.

6.2 Environment Specifications

A more detailed description of each of the environments is given in this section to point out the different aspects tested, which includes:

1. Geometric complexity in terms of the number of objects, polygonal meshes, polygons, and primitives, and the orientation and size (area) of these entities with respect to each other.
2. The distribution of energy in the scene, and the relation of total energy to size (volume occupied) and geometry of the scene.

First, a simple cubic model is shown in *Figure 6.1.* The scene contains 10 objects, 10 meshes, 103 polygons, with 118 unique vertices. The total original energy is 6.525 with a total area of 25.75. The radius of the enclosure is 3.337. This scene contains few occluding objects, a large percentage of emitters to non-emitters (roughly 1/4 of the total area), only planar meshes, emitters with non-zero reflectance, and emitters at right angles to non-emitters.

Secondly, a simple model with primitives is shown in *Figure 6.2.* The scene contains 14 objects, 23 meshes, 714 polygons, and 702 unique vertices. The total original energy is 226.25 with a total area of 804.011. The radius of the enclosure is 11.0494. This scene contains many occluding objects, a small percentage of strong emitters with (zero reflectance) to non-emitters (with high reflectance), non-planar meshes and primitives, and emitters at right angles to non-emitters.

A more complex scene is shown in *Figure 6.3.* The scene contains 26 objects, 63 meshes, 1391 polygons, with 1306 unique vertices. The total original energy is 17801.7, with a total area of 8287.5, and the radius of the enclosure is 44.881. This scene contains many occluding objects consisting of clustered small non-emitters, surrounded by a large mostly empty enclosure consisting of many strong area emitters with zero reflectance. The percentage of primitives to meshes is high (roughly 3 to 1).

A second complex scene, is shown in *Figure 6.4.* The scene contains 41 objects, 160 meshes, 1296 polygons, with 1356 unique vertices. The total original energy is 448.794 with a total area of 2058.14. The radius of the scene is 16.5941. This scene contains very few large area emitters (with zero reflectance) compared to non-emitters. Surfaces do not form an enclosure.

6.3 Results

From statistics gathered for the scenes just mentioned, some results with respect to convergence and CPU utilization, culling to reduce computation time for visibility, and use of adaptive visibility sampling and adaptive subdivision techniques are given in the following sections.

6.3.1 Convergence and Time

Convergence

In terms of convergence (how quickly energy is propagated per iteration), all scenes showed a logarithmic decrease in total energy per iteration. Qualitatively, due to very slow decreasing rates in later iterations roughly the last quarter of the iterations show little discernible change in the image produced. (See *Figure 6.5* for example). Convergence rates were highly dependent on the surface properties and geometry of emitters, and reflectors.

A few strong large area emitters gave steep initial energy drops (e.g. $\sim 50\%$ propagated after 10 iterations for the cubic environment). The presence of an enclosure also greatly affects the rate of convergence, wherein in the absence of one, initial energy drops are steeper and final convergence is reached more quickly. For example, the second complex scene, takes 52 iterations to complete, with about 20% of the original energy left after 12 iterations.

If emitters have small reflectance and are not clustered, as with the cubic scene, convergence is reached quickly, with the energy drop tapering off in later iterations until little energy remains upon completion ($\sim 3\%$). With highly clustered small non-emitters, as in the first complex scene, slower rates of change result in later iterations. In the primitive scene, where a high concentration of strong emitters and strong reflectors exists, as well as increased number of possible shooters due to the presence of tesselated primitives, the convergence rate is quite a bit slower, than the cubic case, with a fair amount of energy left after convergence at 234 iterations (~ 20).

Convergence rates are shown in *Figure A.1 and A.2*

CPU Usage

With respect to the utilization of processing time, the majority of the time per iteration is used for computing form-factors ($\sim$ 85%) of which most of that time is used to compute visibility ($\sim$ 80%). Shaft culling time is comparatively small, taking up about 5% of the total iteration time. Hierarchical bounding volume creation time was mostly dependent on creation of the polygon volume tree (about 90% of the time), with an average insertion time of $\sim 1x10^{-4}$ seconds. Modeling time which includes time to read in data, and create the structure hierarchy (tri-quadtree, vertex-tree, etc.) takes about 0.008 seconds per polygon on average.

6.3.2 Culling

From experimental results, source and receiver culling tends to reduce the number of potential receivers by about 20%, for all scenes tested. Savings in ray visibility testing, and per form-factor time ranged from $\sim$ 23% less time per ray test, and $\sim$ 8% per form-factor for the second complex scene to roughly halving these times for the first complex scene.

The cost of shaft culling is not justified for mostly empty scenes, with few obstructing objects, where the overhead of using a shaft increases the cost per iteration more than any possible savings in ray-intersection time. When shaft culling is appropriate, a ratio-open strategy gives generally better results, unless the scene is mostly empty, in which case a keep-closed strategy may give less overhead time for culling, since percent-overlap ratio testing may be wasted time. Even though more granular shafts are used in this implementation, in contrast to the larger object level shafts used in [Haines92a], similar overall results were obtained.

6.3.3 Visibility Sampling

Ray Casting Time

From experimental scenes used, it seems that those with many occluding objects evenly distributed in the scene will require the least number of ray intersection tests, since once a hit has been found, no further traversal is required. Of the ray intersections found, the majority were with bounding volumes, the percentage of which ranged from 67% for the second complex scene to 97% for the cubic scene. The number of intersections per ray intersection test was quite small for polygons, ranging from intersecting 1% to 0.2% of the time. For bounding volumes, about 60 to 70% of intersection tests resulted in an intersection being found.

Sample Rate

In general, a suggested sampling rate of 4 source samples per receiver vertex, to compute visibility, tended to adjust to 1 sample. Both 1 and 4 sample solutions gave reasonable results with respect to 16 sample per vertex solutions. In terms of convergence, differences from 0.07% (for the cubic scene) to 2% (for the first complex scene) of the total energy left was seen. Time saved is from 12 to 30 percent per iteration, and 22 to 25% per form-factor for roughly 1.2 to 2 times less ray visibility tests.

Adaptive Sampling Rates

Currently, without adjusting the number of samples taken to determine visibility, the increase in cost is roughly 2.5 to 4 times the time per iteration and per form-factor calculation due to about 4 times the number of ray tests, with very small differences in energy left at convergence (less than 1/2% for the first, and less than 1% for the second complex scenes).

6.3.4 Adaptive Subdivision

In the current implementation, the increase in cost associated with using subdivision is quite large, if mutual visibility is maintained between new elements (produced upon subdivision) and the rest of the scene. This was the case with the cubic scene. Costs are reduced for scenes with more occluding objects.

With adjusted sampling rates, 1.4 to 3 times the amount of time per form-factor calculation and roughly a 40% time increase per iteration results to compute 1.4 to 3 times the number of ray tests for 1 level of subdivision allowed. The upper bound was for the mostly empty cubic scene, and the lower bound for both complex scenes. Similar results were obtained from 1 to 2 levels of subdivision for the cubic scene, as for 0 to 1 levels.

Without adjusted sampling, at 1 level compared with no subdivision, increases ranged from about 3 times the amount of time per iteration and 3 times per form-factor to compute about 4 times the number of ray tests for the cubic scene to 1.4, 1.4 and 4 times respectively for the second complex scene.

The increases are mainly due to the lack of form-factor or visibility information reuse and computation of form-factor at vertices. Until reuse is incorporated, tests using minimum area subdivision are inconclusive, using the current restricted subdivision levels.

6.3.5 Images

The first four figures show the initial model geometry and the final results after the radiosity simulation, for the four scenes tested. The next two pictures show an example solution at various stages of completion, and some rendering options in the walk-through program. The last picture shows the results after 200 iterations for a complex scene with 23640 total polygons consisting of 5352 patches, 18288 elements and 17961 vertices; 524 meshes; and 89 objects.

Figure 6.1: Cubic Environment: Shown is simple cubic environment

Figure 6.2: Primitive Environment: Shown is environment after 200 iterations

Figure 6.3: Complex Environment 1: Shown is the first complex environment

Figure 6.4: Complex Environment 2: Shown is the second complex environment

Figure 6.5: Progressive Refinement: Clockwise from the top left, primitive scene at 40, 100, 200 and 234 iterations

Figure 6.6: Walk-through options: Clockwise from top left shows a) Polygonally meshed scene, b) Flat shaded scene, c) Gouraud shaded scene and d) Triangulated Gouraud shaded scene

Figure 6.7: Sample environment: Shown is a more complex environment

Chapter 7

Conclusions and Future Work

In this thesis, a comparative survey has been given which encompasses all major areas of techniques used for computing solutions using the radiosity method. In addition, some of the major advances in each area has been incorporated in a general implementation useful for further experimental work in selection of techniques. From current experimental results, some of the advantages and disadvantages of the techniques implemented have been shown, as well as some of the interdependencies between these techniques. To bring together the various aspects discussed in this paper, this chapter will present some closing remarks with respect to the radiosity techniques examined, and the current implementation presented.

As has been seen, the radiosity method is based on established physically based principles, which allows for realistic image synthesis that does not rely on empirical knowledge. Though the basic principles behind computing energy transport using this method are straight forward, there are a number of difficult choices to be made as to what techniques are suitable when implementing an algorithm for such a method. These choices are further complicated by the fact that many techniques have mutual dependencies.

In the following sections, an outline of a set of suggested techniques, and possible enhancements for the current implementation, are given.

7.1 Radiosity Techniques

First of all, for geometric modeling, current polygonal representations have many problems associated with the fact that polygonal breakdowns cannot capture information that is continuous in nature. Specifically, these representations suffer from inaccurate geometric representations and radiosity gradients, associated rendering problems, cost overhead for maintaining polygonal meshes, and an explicit dependency on these meshes to be able to represent radiosity gradients accurately. Though some techniques have attempted to address these concerns, a preferable choice is to use more continuous representations such as textures, or other parametric descriptions.

In terms of form-factor computation, many methods only compute approximate discrete solutions for visibility and form-factors, or compute continuous solutions which are too computationally expensive for complex scenes with very large numbers of form-factors to compute. Of the current discrete solutions, depth-buffer approaches currently have numerous sampling problems associated with using uniform sampling methods, and insufficient sampling density, which results in inaccurate results. Though the method is computationally fast, due to the usage of graphics hardware, the result is incorrect solutions that are computed more quickly. As such, recent work has concentrated on using ray casting techniques to compute form-factors. To capture diffuse transport, techniques such as Wallace's ([Wallace89]) are suggested, while for capturing more general effects, use of extended form-factors ([Sillion89]) is more suitable. In terms of storage and computation costs, two approaches have stood out. The first is Hanrahans error tolerance method ([Hanrahan91]), which exploits being able to share information between surface subdivision levels, and works best for scenes which are polygonalized. The second approach is to use variance-reducing probabilistic methods, which are suitable for non-polygonized scenes as well as polygonized ones, and have less dependency on grouping in order to produce reduced sampling costs. This second approach still has problems with error reduction, sufficient sampling to capture energy gradients at surfaces, and is restricted due to it's computational expense.

When performing sampling to compute visibility and energy gradients, most methods are incomplete as they do not consider all important factors. For methods to be complete they must consider distributions of energy at the surface sampling from, and distributions at surfaces being sampled, as well as the geometry of these surfaces. To date the most complete methods include Hanrahan's BF-refinement method ([Hanrahan91]), and Haines shaft culling method ([Haines92]) in terms of capturing diffuse transport. For specular transport, current methods are still lacking in terms of sample reuse.

To capture or store gradients found, adaptive storage methods introduced by Cohen ([Cohen86]) are preferable for computation and storage cost reduction. In terms of capturing general radiative properties, methods of He and Sillion ([He91] and [Sillion91] respectively) incorporate the most accurate physically based illumination models.

For capturing global illumination, radiosity solutions that incorporate progressive refinement ([Chen89]) are preferable as they allow reduced storage costs, and incremental feedback. To incorporate diffuse and specular effects efficiently, hybrid multi-pass methods, such as [Chen91] which utilize both ray tracing and radiosity based solutions, are preferable, as each method has it's own advantage with respect to the transport mechanisms that may be simulated. As with sampling methods, there is still little shared information between different passes in current implementations.

Even with speed improvements, as with the case of depth-buffer form-factor techniques, fast solutions may not be accurate ones. Currently, there is a lack of published general criteria to measure physical accuracy of solutions for scenes with general radiative and geometric properties, though such research is currently underway ([Greenberg91]). As well, measures of "quality" of solutions are incomplete, as many current algorithms only use a single measure of quality, which may not be complete or generally applicable.

Assuming solutions are accurate, accuracy and minimal cost for storage of results (i.e. storage of radiosity distributions) is desired. Sillion's ([Sillion91]) storage scheme has shown useful characteristics of roughly fixed storage cost regardless of the complexity of the energy distribution stored, and efficient placement and movement of distributions. For the most flexible placement scheme, and usage of the real geometry of objects, storage at texture map locations is preferable to placement at polygon or vertex points in polygonal representations. Further work is still required to derive texture map representations that avoid polygonization and uniform subdivision in such maps.

For capturing more than just ideal diffuse transport, rendering using Gouraud shading is not a practical option. Thus, current techniques that use ray tracing methods are preferable for capturing general transport. Reconstruction of gradients still poses problems, due to insufficient samples for proper reconstruction, simplistic reconstruction methods, and insufficient research into examining accountability for samples at surfaces. From the set of available techniques, a choice still must be made between accuracy and flexibility versus speed.

When considering dynamic environments, the current choice has been speed over accuracy, as all current solutions only compute ideal diffuse transport using depth-buffer techniques. Important advances have been incremental form-factor ([Chen90]), and incremental radiosity ([Sillion89]) computations which exploit temporal coherence. In terms of maintaining accurate solutions when geometric or radiative changes occur in an environment, the criteria for determining the order of propagating energy is still incomplete. In general, two important principles have been used: that of smooth image refinement ([Chen89]), and consideration of properties important to the viewer. This second point includes consideration of viewing position ([Baum90] [Heckbert90]), and specular effects, and texturing deemed important to the viewer ([Chen91]).

Finally, current parallel solutions have also concentrated on speed versus accuracy. Currently lacking are implementations for hybrid ray-tracing / radiosity methods, adaptive subdivision and sampling techniques, and form-factors and energy transport that incorporates generalized radiative properties.

7.2 Radiosity Implementation

The current implementation presented in this paper has chosen techniques based on the restrictions imposed by the radiative and geometric properties associated with environments best suited for a non-parallel progressive refinement radiosity algorithm, designed for machine-independence, and suitable for real-time scene walk-throughs. The current major restrictions are the assumption of ideal diffuse radiative properties and use of polygonized environments. As the type of environments that maintain these assumptions is quite small, a natural extension would be the use of techniques that relax these assumptions. Among the enhancements would be the use of form-factors computed using ray tracing (extended form-factors), as opposed to ray casting and the use of textures versus the use of polygonal meshes for radiosity storage and sampling. This would result in extensions being required for input parameters for models, and rendering using a ray tracing method. For input models, in addition to specifications for more general radiative properties, incorporation of many of modeling constraints found in [Baum91] are still useful. For rendering, though there are many publicly available ray tracing programs available, enhancements are required to incorporate energy distributions calculated using the radiosity simulation into energy transport computed using ray tracing, as seen in [Sillion91]. To capture these distributions, better storage mechanisms are required such as that of [Sillion91]. Finally, better reuse of visibility and form factor information, and information found in the structure hierarchy is required for efficient computation and storage. To this end, sample reuse from [Wallace89] and [Campbell89], and BF-refinement from [Hanrahan91] for reduced form-factor interactions are desirable. Currently, work is underway into incorporating BF-refinement [Hanrahan91], use of modeling constraints in [Baum91], and sample reuse.

Appendix A

Statistics

Notation

The following notation is used for tables of statistics given below.

Parameter	Description
Tot iter	Total number of iterations executed, either before convergence, or before reaching 200 iterations.
E left	Energy left to propagate in the scene. Assuming the dimensions of the scenes modeled are given in metres, energy is specified in units of watts.
Perc left	Percentage of original energy left in the scene.
PR	Total time to compute radiosity solution.
HBV	Preprocessing time to compute object and polygon bounding volumes.
Avg Iter	Average computation time per iteration.
Rec / Elem	The ratio of receivers to elements for the complete solution.
Avg ff	Average computation time for form-factor computation per iteration.
Avg shaft	Average computation time for shaft culling per iteration.
Avg ray	Average computation time for ray visibility testing per form-factor per iteration.
x	x = Cone, Cube, Cyl, Sph, or Poly, which represents either a cone, cube, cylinder, sphere or polygon respectively.
Int / ray	Average number of total intersections per ray test.
Ix / ray	Average number of intersections with x per ray test.
Tx / ray	Average number of intersections tests with x per ray test.
Ix / RT	Average number of intersections with x per ray test.
Str	Shaft culling strategy where KC, AO, OO, and RO represent "Keep closed", "Always open", "Overlap open" and 40% "ratio open".
Boxes T	Number of bounding boxes tested in shaft culling.
In / Out / Over / Cont	Number of boxes inside, outside, overlapping, or containing the shaft tested.
TotC	Total number of candidates found in shaft culling.
C/Shaft	Average number of candidates per shaft per iteration.

A.1 Convergence Graphs

The following graphs shows the percentage of total energy along the vertical axis versus the number of iterations along the horizontal axis.

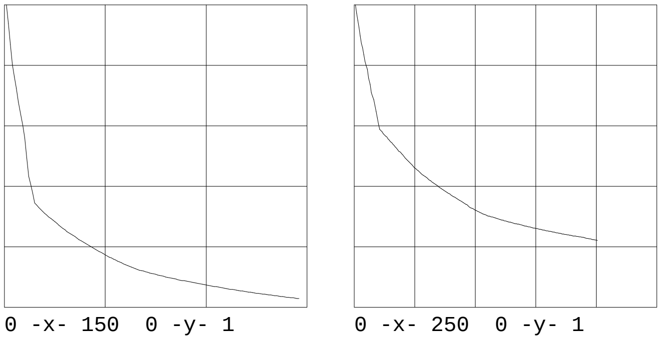

Figure A.1: Convergence rates: Cubic environment on the left, and primitive environment on the right

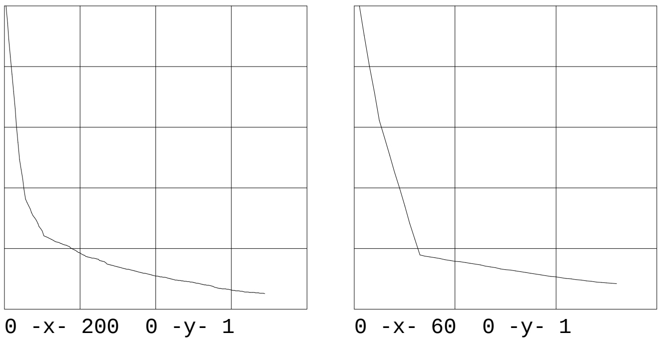

Figure A.2: Convergence rates: Complex environment 1 on the left and 2 on the right

A.2 Tables of statistics

For the proceeding tables of statistics, each column indicates a different set of test conditions, given in Chapter 6, and each row indicates algorithm parameters measured. All computation are in seconds.

A.2.1 Tables for Cubic Scene

Preprocess time for bounding volume tree creation is ~ 0.03 seconds, and 0.58 seconds for model creation.

Cubic	WALK	M0R4S0A1	M0R4S1A1	M0R4S2A1	M0R4S3A1	M0R1S3A1	M0R4S3A0
E Left	0.182	0.1773	0.1773	0.1773	0.1773	0.1773	0.18122
Perc. Left	2.792	2.717	2.717	2.717	2.717	2.717	2.777
PR	143.58	128.79	132.3	129.7	126.34	126.45	389.15
Avg Iter	0.983	0.882	0.906	0.888	0.865	0.866	2.612
Avg ff	0.706	0.606	0.603	0.591	0.590	0.590	2.33
Avg shaft	0.081	0.082	0.105	0.101	0.081	.0842	0.0851
Avg Ray	0.6258	0.5345	0.5314	0.5248	0.5213	0.5154	2.09409

Table A.1: Computation time: The total number of iterations is 146.

Cubic	WALK	M0R4S0A1	M0R4S1A1	M0R4S2A1	M0R4S3A1	M0R1S3A1	M0R4S3A0
Int / ray	10.3911	10.484	10.39	10.288	10.44	10.44	10.57
Int / BV	10.121	10.204	10.105	10.007	10.16	10.16	10.29
IPoly / ray	0.1351	0.1402	0.1402	0.1402	0.1402	0.1402	0.139
TBV / ray	17.0328	17.038	16.914	16.771	16.995	16.995	17.07
TPoly / ray	16.880	17.167	17.427	17.223	16.817	16.817	17.115
IBV / RT	0.594	0.599	0.598	0.597	0.598	0.598	0.603
IPoly / RT	0.0082	0.0082	0.0082	0.0082	0.0084	0.0084	0.0081

Table A.2: Ray statistics: At 16 samples per vertex 50584 rays are shot, at 1 and 4 samples per vertex: 42052, and 172128 rays for M0R4S4A0.

Cubic	WALK	M0R4S0A1	M0R4S1A1	M0R4S2A1	M0R4S3A1	M0R1S3A1	M0R4S3A0
Str	RO	KC	AO	OO	R0	RO	RO
Boxes T	90649	90649	104465	104409	90649	90649	92824
In	4	4	117	91	4	4	4
Out	67915	67915	73453	73453	67915	67195	69530
Over	1296	1296	4242	4456	1296	1296	1352
Cont	9836	9836	9925	9836	9836	9836	10053
TotC	11136	11136	14284	14383	11136	11136	11409
C/shaft	76.274	76.274	97.836	98.514	76.274	76.274	76.571

Table A.3: Shaft statistics

Cubic	M1R4S3A0	M2R4S3A0	M1R4S3A0	M1R4S3A0a5
Tot CPU	154	157	151	151
E Left	0.181	0.189	0.185	0.185
Perc. Left	2.776	2.890	2.832	2.832
PR	1460.25	5086.87	1759.77	1766.6
Avg Iter	9.482	32.4	11.654	11.699
Avg ff	8.356	28.357	7.368	7.398
Avg shaft	0.315	1.092	1.103	1.1
Avg Ray	7.4215	25.233	6.442	6.467

Table A.4: Computation time

Cubic	M1R4S3A0	M2R4S3A0	M1R4S3A0	M1R4S3A0a5
Tot rays	672432	2371952	581796	581796
Int / ray	9.931	9.685	9.631	9.631
Int / BV	9.6365	9.638	9.338	9.338
IPoly / ray	0.1472	0.150	0.146	0.146
TBV / ray	16.692	16.534	16.546	16.546
TPoly / ray	15.494	14.748	14.596	14.596
IBV / RT	0.577	0.568	0.564	0.564
IPoly / RT	0.01	0.01	0.01	0.01

Table A.5: Ray casting statistics

Cubic	M1R4S3A0	M2R4S3A0	M1R4S3A0	M1R4S3A0a5
Boxes T	358992	1263828	1238301	1238301
In	0	0	0	0
Out	271747	960806	941205	941205
Over	2571	5524	5266	5266
Cont	40308	143864	141244	141244
TotC	42879	149388	146510	146510
C/shaft	278.44	951.52	970.26	970.26

Table A.6: Shaft statistics: A 40 percent ratio open strategy was used.

A.2.2 Tables for Primitive Scene

Preprocess time for hierarchical bounding volume building is ~ 0.035 seconds, and ~ 6.03 seconds for model creation.

Prim	WALK	M0R4S0A1	M0R4S1A1	M0R4S2A1	M0R4S3A1	M0R1S3A1	M0R4S3A0
E Left	61.761	56.61	56.61	56.61	56.61	56.61	66.476
Perc. Left	27.298	24.779	24.779	24.779	24.779	24.779	29.382
PR	895.17	819.78	847.08	845.67	820.06	820	1560.21
Avg Iter	4.453	4.078	4.214	4.207	4.08	4.079	7.762
Avg ff	1.708	1.329	1.313	1.307	1.329	1.314	5.008
Avg shaft	1.09	1.09	1.25	1.25	1.088	1.098	1.088
Avg Ray	1.011	1.089	0.786	0.774	0.77	0.759	3.031

Table A.7: Computation time: Total number of iterations is 200. Number of receivers per elements is ~ 84.13 percent.

Prim	WALK	M0R4S0A1	M0R4S1A1	M0R4S2A1	M0R4S3A1	M0R1S3A1	M0R4S3A0
Int / ray	1.4761	1.544	1.388	1.374	1.479	1.479	1.51
Int / BV	0.977	1.03	0.886	0.875	0.9833	0.9833	1.01
ICube / ray	0.135	0.142	0.142	0.142	0.15	0.15	0.165
ICone / ray	0.119	0.102	0.102	0.102	0.103	0.103	0.102
ICyl / ray	0.141	0.152	0.153	0.152	0.151	0.15	0.145
ISph / ray	0.104	0.104	0.104	0.104	0.103	0.103	0.103
IPoly / ray	0.022	0.031	0.029	0.029	0.02	0.02	0.02
TBV / ray	3.326	3.22	3	2.98	3.168	3.168	3.153
TCube / ray	0.326	0.337	0.34	0.337	0.344	0.344	0.341
TCone / ray	0.571	0.552	0.559	0.552	0.558	0.558	0.551
TCyl / ray	0.439	0.446	0.448	0.446	0.443	0.443	0.438
TSph / ray	0.499	0.482	0.486	0.4816	0.481	0.481	0.472
TPoly / ray	1.06	1.338	1.447	1.4154	1.12	1.12	1.14
IBV / RT	0.294	0.32	0.295	0.2937	0.3103	0.3104	0.321
ICube / RT	0.417	0.42	0.419	0.4204	0.4365	0.4365	0.4838
ICone / RT	0.210	0.184	0.183	0.1845	0.1838	0.1838	0.1843
ICyl / RT	0.321	0.340	0.342	0.3402	0.3405	0.3405	0.3297
ISph / RT	0.208	0.215	0.215	0.2157	0.2137	0.2137	0.2172
IPoly / RT	0.021	0.023	0.02	0.02	0.018	0.018	0.021

Table A.8: Ray statistics: At 16 samples per vertex 649844 rays are shot, at 4 samples per vertex: 481784, and 1930816 rays for M0R4S4A0.

Prim	WALK	M0R4S0A1	M0R4S1A1	M0R4S2A1	M0R4S3A1	M0R1S3A1	M0R4S3A0
Str	RO	KC	AO	OO	R0	RO	RO
Boxes T	1409286	1397739	1553729	1554911	1402013	1402013	1402125
In	238	241	523	468	240	240	233
Out	986557	979213	1043144	1043651	980958	980958	981082
Over	170327	167827	198037	203116	169889	169889	169884
Cont	33335	33244	33801	33244	33104	33104	32920
TotC	203900	201312	232361	236828	203233	203233	203037
C/shaft	1019.5	1006.56	1161.81	1184.14	1016.16	1016.16	1015.18

Table A.9: Shaft statistics

Prim	M1R4S3A0	M1R4S3A1a5	M1R4S3A1
E Left	68.04	56.82	56.51
Perc. Left	30.07	25.11	24.98
PR	4275.91	2249.39	2334.1
Avg Iter	21.273	11.191	11.612
Avg ff	14.111	3.774	4.045
Avg shaft	2.921	2.945	3.588
Avg Ray	8.797	2.242	2.52

Table A.10: Computation time

Prim	M1R4S3A0	M1R4S3A1a5	M1R4S3A1
Tot rays	5197288	1315980	1316636
Int / ray	1.764	1.733	1.734
Int / BV	1.185	1.157	1.158
ICube / ray	0.161	0.147	0.158
ICone / ray	0.105	0.104	0.104
ICyl / ray	0.2061	0.2168	0.2169
ISph / ray	0.119	0.1169	0.1168
IPoly / ray	0.0243	0.0213	0.0213
TBV / ray	3.305	3.30	3.303
TCube / ray	0.3099	0.3068	0.3068
TCone / ray	0.5199	0.5236	0.524
TCyl / ray	0.4553	0.4664	0.4665
TSph / ray	0.4674	0.4701	0.4702
TPoly / ray	1.4589	1.4397	1.441
IBV / RT	0.3587	0.3505	0.3507
ICube / RT	0.52	0.4791	0.4797
ICone / RT	0.2024	0.1985	0.1984
ICyl / RT	0.4525	0.4649	0.4652
ISph / RT	0.2546	0.2487	0.2485
IPoly / RT	0.0167	0.0148	0.0148

Table A.11: Ray casting statistics

Prim	M1R4S3A0	M1R4S3A1a5	M1R4S3A1
Boxes T	3723961	3787722	3789150
In	180	175	174
Out	2595156	2641611	2642486
Over	460406	469449	469675
Cont	87382	88384	88491
TotC	547968	558008	558340
C/shaft	2739.84	2790.04	2791.7

Table A.12: Shaft statistics: A 40 percent ratio open strategy was used.

A.2.3 Tables for Complex Scene 1

Time for bounding volume tree creation is ~ 0.19 seconds, and ~ 12.68 seconds for model creation.

Compl1	WALK	M0R4S0A1	M0R4S1A1	M0R4S2A1	M0R4S3A1	M0R1S3A1	M0R4S3A0
Iter	172	166	166	166	166	166	173
E Left	427.032	431.4	431.4	431.4	431.4	431.4	438.854
Perc. Left	5.153	5.21	5.21	5.21	5.21	5.21	5.29
PR	2755.69	2414.64	2507.77	2512.88	2276.25	2275.77	5781.08
Avg Iter	16.02	14.55	15.11	15.14	13.63	13.626	33.416
Rec / Elem	0.842	0.838	0.838	0.838	0.838	0.838	0.843
Avg ff	9.364	7.921	7.03	7.07	7.0	7.02	26.77
Avg shaft	3.492	3.46	4.923	4.924	3.48	3.47	3.481
Avg Ray	7.992	6.867	5.95	5.98	5.922	5.928	22.982

Table A.13: Computation time

Compl1	WALK	M0R4S0A1	M0R4S1A1	M0R4S2A1	M0R4S3A1	M0R1S3A1	M0R4S3A0
Tot rays	1047992	745172	745172	745172	745172	745172	3126672
Int / ray	7.166	7.314	6.2	6.32	6.17	6.17	7.07
Int / BV	6.78	6.864	5.79	5.91	5.78	5.78	6.68
ICube / ray	0.495	0.5	0.5	0.5	0.5	0.5	0.493
ICone / ray	0.0319	0.024	0.024	0.024	0.024	0.024	0.026
ISph / ray	0.152	0.151	0.151	0.151	0.151	0.151	0.152
IPoly / ray	0.0338	0.06	0.05	0.05	0.034	0.034	0.033
TBV / ray	11.349	11.483	9.54	9.80	11.487	11.487	11.368
TCube / ray	1.7	1.67	1.67	1.67	1.67	1.67	1.674
TCone / ray	0.203	0.194	0.194	0.194	0.194	0.194	0.196
TSph / ray	0.923	0.916	0.916	0.916	0.916	0.916	0.899
TPoly / ray	8.33	11.1633	9.748	9.726	8.21	8.21	7.929
IBV / RT	0.597	0.598	0.607	0.603	0.59	0.59	0.587
ICube / RT	0.291	0.3	0.3	0.3	0.3	0.3	0.295
ICone / RT	0.157	0.121	0.121	0.121	0.121	0.121	0.133
ISph / RT	0.165	0.165	0.165	0.165	0.165	0.165	0.169
IPoly / RT	0.004	0.006	0.006	0.006	0.004	0.004	0.004

Table A.14: Ray statistics

Compl1	WALK	M0R4S0A1	M0R4S1A1	M0R4S2A1	M0R4S3A1	M0R1S3A1	M0R4S3A0
Str	RO	KC	AO	OO	RO	RO	RO
Boxes T	3407158	3266304	4437292	4430643	3291581	3291581	3430251
In	3714	3569	12204	9219	3597	3597	3733
Out	1000453	956445	1509552	1508493	965801	965801	1007066
Over	387728	372240	714080	726339	374600	374600	390687
Cont	171895	164932	16507	164932	166241	166241	172966
TotC	563337	540741	891358	900490	544438	544438	567386
C/shaft	3275.22	3257.48	5369.63	5424.64	3260.11	3260.11	3279.69

Table A.15: Shaft statistics

Compl1	M1R4S3A0	M1R4S3A1a5	M1R4S3A1
Iter	172	177	177
E Left	438.854	453.469	453.469
Perc. Left	5.29	5.472	5.472
PR	5791.08	3343.3	3357.75
Avg Iter	33.416	18.89	18.97
Rec / Elem	0.843	0.83	0.83
Avg ff	26.774	9.775	9.71
Avg shaft	3.481	4.899	4.819
Avg Ray	22.982	8.253	8.27

Table A.16: Computation times and convergence

Compl1	M1R4S3A0	M1R4S3A1a5	M1R4S3A1
Tot rays	3126672	1112468	1112468
Int / ray	7.072	7.228	7.228
Int / BV	6.678	6.878	6.878
ICube / ray	0.493	0.5	0.5
ICone / ray	0.026	0.019	0.019
ISph / ray	0.152	0.111	0.111
IPoly / ray	0.033	0.033	0.033
TBV / ray	11.368	11.502	11.502
TCube / ray	1.674	1.715	1.715
TCone / ray	0.196	0.153	0.153
TSph / ray	0.899	0.826	0.826
TPoly / ray	7.939	8.148	8.148
IBV / RT	0.587	0.598	0.598
ICube / RT	0.295	0.292	0.292
ICone / RT	0.133	0.124	0.124
ISph / RT	0.169	0.135	0.135
IPoly / RT	0.004	0.004	0.004

Table A.17: Ray casting statistics

Compl1	M1R4S3A0	M1R4S3A1a5	M1R4S3A1
Boxes T	3430251	4832904	4832904
In	3733	4667	4667
Out	1007066	1468601	1468601
Over	390687	501824	501824
Cont	172966	243643	243643
TotC	567386	750134	750134
C/shaft	3279.69	4238.05	4238.05

Table A.18: Shaft statistics: A 40 percent ratio open strategy was used.

A.2.4 Tables for Complex Scene 2

Time for hierarchical bounding volume creation is ~ 0.24 seconds, and ~ 10.14 for model creation.

Compl2	WALK	M0R4S0A1	M0R4S1A1	M0R4S2A1	M0R4S3A1	M0R1S3A1	M0R4S3A0
E Left	36.812	35.008	35.008	34.515	34.86	34.86	38.858
Perc. Left	8.202	7.8	7.8	7.691	7.768	7.768 7.768	8.658
PR	589.66	458.53	477.66	473.63	456	456.79	1174.74
Avg Iter	11.335	8.813	9.181	9.104	8.765	8.78	22.587
Avg ff	7.772	5.262	5.244	5.258	5.238	5.236	19.075
Avg shaft	0.919	0.929	1.297	1.185	0.8925	0.909	0.895
Avg Ray	6.717	4.579	4.595	4.608	4.601	4.574	16.8425

Table A.19: Computation times: Total number of iterations is 52 for each run. The average number of receivers per elements per iteration is 0.504.

Compl2	WALK	M0R4S0A1	M0R4S1A1	M0R4S2A1	M0R4S3A1	M0R1S3A1	M0R4S3A0
Tot rays	242784	134112	134112	134112	134112	134112	536448
Int / ray	7.89	9.992	9.692	9.769	9.918	9.918	9.271
Int / BV	7.778	9.844	9.541	9.618	9.767	9.767	9.14
ICube / ray	0.247	0.312	0.317	0.319	0.317	0.317	0.263
ICyl / ray	0.014	0.014	0.0129	0.015	0.0151	0.0151	0.01
IPoly / ray	0.016	0.019	0.021	0.02	0.019	0.019	0.017
TBV / ray	11.103	13.575	13.179	13.299	13.478	13.478	12.98
TCube / ray	1.773	2.260	2.199	2.247	2.258	2.258	2.102
TCyl / ray	0.069	0.08	0.073	0.081	0.08	0.08	0.053
TPoly / ray	10.491	13.424	13.618	13.446	13.195	13.195	12.198
IBV / RT	0.701	0.725	0.724	0.723	0.725	0.725	0.704
ICube / RT	0.139	0.138	0.144	0.142	0.141	0.141	0.125
ICyl / RT	0.207	0.184	0.177	0.185	0.191	0.191	0.198
IPoly / RT	0.002	0.001	0.002	0.001	0.001	0.001	0.001

Table A.20: Ray statistics

Compl2	WALK	M0R4S0A1	M0R4S1A1	M0R4S2A1	M0R4S3A1	M0R1S3A1	M0R4S3A0
Str	RO	KC	AO	OO	R0	RO	RO
Boxes T	194593	184220	291723	264891	194593	194593	194593
In	222	203	10089	2656	222	222	222
Out	80883	73097	119712	112265	80883	80883	80883
Over	10845	8992	29076	26824	10845	10845	10845
Cont	32660	32660	32811	32660	32660	32660	32660
TotC	43727	41855	71976	62140	43727	43727	43727
C/shaft	840.904	804.904	1384.15	1195	840.904	840.904	840.904

Table A.21: Shaft statistics

Compl2	M1R4S3A0	M1R4S3A1a5	M1R4S3A1
E Left	37.537	33.684	33.684
Perc. Left	8.364	7.505	7.505
PR	1607.6	636.12	645.82
Avg Iter	30.911	12.229	12.415
Avg ff	25.884	7.02	7.02
Avg shaft	1.327	1.462	1.626
Avg Ray	22.769	6.04	5.98

Table A.22: Computation time. Average number of receivers per elements is 0.509.

Compl2	M1R4S3A0	M1R4S3A1a5	M1R4S3A1
Tot rays	751240	189720	189720
Int / ray	8.819	9.156	9.156
Int / BV	8.679	9.003	9.003
ICube / ray	0.293	0.345	0.345
ICyl / ray	0.008	0.012	0.012
IPoly / ray	0.021	0.021	0.021
TBV / ray	12.418	12.579	12.579
TCube / ray	2.045	2.115	2.115
TCyl / ray	0.051	0.069	0.069
TPoly / ray	11.658	12.164	12.164
IBV / RT	0.699	0.716	0.716
ICube / RT	0.143	0.163	0.163
ICyl / RT	0.164	0.169	0.169
IPoly / RT	0.002	0.002	0.002

Table A.23: Ray casting statistics

Compl2	M1R4S3A0	M1R4S3A1a5	M1R4S3A1
Boxes T	281703	283872	283872
In	353	360	360
Out	119635	120284	120284
Over	17206	17317	17317
Cont	45805	46330	46330
TotC	63364	64007	64007
C/shaft	1218.54	1230.9	1230.9

Table A.24: Shaft statistics: A 40 percent ratio open strategy was used.

Appendix B

Glossary

This appendix provides brief descriptions of various terms mentioned in the text of this document. For more detailed explanations the interested reader may refer to general references such as [Foley90] and [Hall89].

- **A-buffer** : Method to antialias, by using a subpixel viewing grid.
- **Aliasing** : Resulting misrepresentations due to sampling with regularly spaced samples at a rate less than the Nyquist rate.
- **Ambient term** : Empirical constant used to represent illumination from all directions on a surface.
- **Back-face Culling** : Eliminating from consideration, all faces of an object whose surface normals point away from a given viewing position.
- **Beam tracing** : Tracing of polygonal beams versus single rays when ray tracing.
- **Bilinear interpolation** : Interpolation in two dimensions that a surface is defined in.
- **Bilinear subdivision** : Subdivision in two dimensions that a surface is defined in.
- **Binary space partitioning tree** : Tree with inner nodes representing partitioning planes and leaves representing regions of space.
- **Depth buffering** : Method to compute visibility by scan converting and sorting projections of polygons by their depth values, with respect to a viewing position, in a pixel level buffer called a **depth buffer**. An **item buffer** is a pixel level buffer that stores the corresponding identifier of polygons that are visible in the pixels of the depth buffer.
- **Distributed ray tracing** : Stochastically distributing rays in space to sample quantities.
- **Double-buffering**: Process of using two buffers, such that one buffer displays the current image, while the other is written to. Once finished writing, the roles of the buffers switch.
- **Extrusion** : Surface formed by translating a 2d curve through space.

- **Filtering** : Using weighted sampling to reconstruct a signal.
- **Gouraud shading** : intensity or colour interpolation between vertex values of polygons projected into image space.
- **Image space** : The coordinate system of the viewing screen.
- **Importance sampling** : Using proportionally more samples at locations of higher weight.
- **Light ray tracing** : Ray tracing from position on light sources into a given scene.
- **Model space** : Coordinate system associated with a model.
- **Non-uniform sampling** : Sampling in a non-uniform distribution.
- **Nyquist Rate** : Minimum sampling rate of two times the highest frequency component in the spectrum of a signal in order to reconstruct the signal from it's samples without aliasing.
- **Quadtree** : Tree produced by successively subdividing a 2D plane in both dimensions to form quadrants. An **octree** is the three-dimensional analog to a quadtree.
- **Perspective distortion** : Anomalies caused by interpolating data after a perspective transformation versus interpolating in world space.
- **Perspective warp** : Use of a series of image transformations to simulate the effect of a perspective transformation of an image.
- **Primitive instancing** : Reuse of a parameterized original object description.
- **Render pipeline** : Implementation in software and/or hardware of a display process that maps a model to an image. The process if referred to as **rendering**.
- **Rotation variant** : Change in properties of the image of a surface as this surface is rotated in space.
- **RMS** : Root mean squared.
- **Shadow rays** : In a ray tracing method, these are rays shot from a point on a surface to the lights in the scene to determine visibility of the point to the lights.
- **Summed area tables** : Method to find the integral between two points on an 2D image by subtracting the horizontal and vertical areas between the minimum and maximum of these points in each dimension, and adding back the intersecting area of the subtracted areas.
- **Surface detail polygon** : Coplanar polygon associated with a polygon in a model, such that properties of the detail polygon are used versus the properties of the model polygon, when computing quantities.
- **Surface of revolution** : Surface formed by rotating a 2d curve about some axis.
- **Texture mapping** : mapping a pattern to a surface.
- **Tri-quadtree** : Tree formed by successive bilinear subdivision of quadrilaterals or triangles to form quadrants.

- **Warnock area subdivision** : Exploitation of area coherence, by subdividing a grid until projected polygons are either inside, outside, overlap or surround a grid cell.
- **Weiler-Atherton algorithm** : Visibility algorithm that finds areas of a model shadowed from each of the lights in a scene, then mapping these shadowed areas onto the model, before rendering from a given viewing position.
- **Winged-edge data structure** : Construct to provide adjacency information, such that constant time is required for searching for vertices and faces with respect to an edge in a boundary representation.

Appendix C

Example Usage

This appendix provides a brief example of the usage of some techniques described in *Chapter 4*. Let us consider a hollow cubic enclosure with a small sphere centered within the cube. The enclosure contains no participating media, and all surfaces are opaque. An outline of the steps required to compute a progressive refinement solution which uses the hemicube method to compute form factors, and Gouraud shading to render images follows:

1. Assume that the normals of the sphere face outward, and the normals of the cube face inwards towards space of the enclosure.
2. Assign radiative properties to the cube and sphere, including emission and reflectance values. Let us assume that one face of the cube has non-zero emission, and all other faces of the cube, and the sphere have zero emission, but non-zero reflectance values.
3. Polygonize the cube and sphere to the desired level of subdivision, making sure to maintain proper modeling constraints.
4. Patches in the scene are sorted by total energy, and the shooter is chosen. The choice of the first shooting patch would be a patch belonging to the emitting face of the cube.
5. Form an imaginary hemicube centered at the front face of the shooter. Project all surfaces onto each of the faces of the hemicube, and compute form factors values.
6. Compute, and propagate the radiosity from the shooter to all surfaces visible from the center of the shooter.
7. Subdivide the shooter based on source subdivision criteria as required. Perform recomputation of form-factors as required.
8. Subdivide receiver patches based on receiver subdivision criteria (e.g. form-factor and radiosity gradients) as required. Perform recomputation of form-factors as required.
9. Patches in the scene are sorted by total energy, and the next shooter is chosen.
10. Go back to step 5, and repeat until the energy per shooter is less than some user defined tolerance.
11. Compute vertex radiosities by interpolating patch radiosity values.
12. Choose a position to view the scene from, and use Gouraud shading to render the scene.
13. Store form-factor and radiosity values if desired.

Bibliography

[Airey89] Airey, John M., M. Ouh-young, "Two Adaptive Techniques Let Progressive Radiosity Outperform the Traditional Radiosity Algorithm", Technical Report TR89-020, University of North Carolina Department of Computer Science, 1989.

[Airey90] Airey, John M., John H. Rohlf, Frederick P. Brooks, Jr., "Towards Image Realism with Interactive Update Rates in Complex Virtual Building Environments", *Computer Graphics (1990 Symposium on Interactive 3D Graphics)*, vol. 24, no. 2, pp. 41-50, March 1990.

[Alias89] Alias Research Inc., *QuickModel (Version 1.0)* modeling program, 1989.

[Amana87] Amanatides, John, Andrew Woo, *Optik (Version 1.2a)* ray tracing program, Dynamic Graphics Project, University of Toronto, 1987.

[Arvo86] Arvo, James, "Backward Ray Tracing", *SIGGRAPH '86 Developments in Ray Tracing seminar notes*, vol. 12, (also appeared in SIGGRAPH '89 Radiosity course notes), Aug. 1986.

[Arvo89] Arvo, James, *An Introduction to Ray Tracing*, Academic Press, Boston, 1989.

[Arvo91] Arvo, James, Andrew S. Glasner, et. al., *Graphics Gems II*, Academic Press, Boston, 1991.

[Blinn76] Blinn, James F., Martin E. Newell, "Texture and Reflection in Computer-Generated Images", *Communications on the ACM*, vol. 19, no. 10, pp. 542-547, Oct. 1976.

[Blinn78] Blinn, James F., "Simulation of Wrinkled Surfaces", *Computer Graphics (SIGGRAPH '78 Proceedings)*, vol. 7, no. 1, pp. 286-292, July 1978.

[Baum86] Baum, Daniel R., John R. Wallace, Michael F. Cohen, Donald P. Greenberg, "The Back-Buffer Algorithm: An Extension of the Radiosity Method to Dynamic Environments", *Visual Computer*, vol. 2, pp. 298-306, 1986.

[Baum89] Baum, Daniel R., Holly E. Rushmeier, James M. Winget, "Improving Radiosity Solutions Through the Use of Analytically Determined Form-Factors", *Computer Graphics (SIGGRAPH '89 Proceedings)*, vol. 23, no. 3, pp. 325-334, July 1989.

[Baum90] Baum, Daniel R., James M. Winget, "Real Time Radiosity Through Parallel Processing and Hardware Acceleration", *Computer Graphics (1990 Symposium on Interactive 3D Graphics)*, vol. 24, no. 2, pp. 67-75, March 1990.

[Baum91] Baum, Daniel R., James M. Winget, Stephan Mann, Kevin P. Smith, "Making Radiosity Usable: Automatic Preprocessing and Meshing Techniques for the Generation of Accurate Radiosity Solutions", *Computer Graphics (SIGGRAPH '91 Proceedings)*, vol. 25, no. 4, pp. 51-60, August 1991.

[Beran91] Beran, Jeffrey C., Koehn, Mark J., "A Cubic Tetrahedral Adaptation of the Hemi-cube Algorithm", in *Graphics Gems II*, James Arvo ed., Boston, 1991.

[Bergman85] Bergman, Larry, Henry Fuches, Eric Grant, Susan Spach, "Image Rendering by Adaptive Refinement", *Computer Graphics (SIGGRAPH '86 Proceedings)*, vol. 19, no. 3, pp. 31-40, July, 1985.

[Bu89] Bu, Jichun, E.F. Deprettre, "A VLSI system architecture for high-speed radiative transfer 3D image synthesis", *Visual Computer*, vol. 5(3), pp. 121-133, June 1989.

[Buck] David Buck, *DKBTrace*, University of Carleton.

[Buckalew89] Buckalew, C., Donald S. Fussell, "Illumination Networks: Fast Realistic Rendering with General Reflectance Functions", *Computer Graphics (SIGGRAPH '89 Proceedings)*, vol. 23, no. 3, pp. 89-98, 1989.

[Campbell90] Campbell, III, A.T., Donald S. Fussell, "Adaptive Mesh Generation for Global Diffuse Illumination", *Computer Graphics (SIGGRAPH '90 Proceedings)*, vol. 24, no. 4, pp. 155-164, August, 1990.

[Chatt87] Chattopadhyay, Subdeb, Akira Fujimoto, "Bi-directional Ray Tracing", *Computer Graphics 1987 (Proceedings of CG International '87)*, Tosiyasu Kunii ed., Springer Verlag, Tokyo, pp. 335-343, 1987.

[Chen89] Chen, Shenchang Eric, "A Progressive Radiosity Method and its Implementation in a Distributed Processing Environment", Master's Thesis, Program of Computer Graphics, Cornell University, January 1989.

[Chen90] Chen, Shenchang Eric, "Incremental Radiosity: An Extension of Progressive Radiosity to an Interactive Image Synthesis System", *Computer Graphics (SIGGRAPH '90 Proceedings)*, vol. 24, no. 4, pp. 134-144, August, 1990.

[Chen91] Chen, Shenchang Eric, Holly Rushmeier, Gavin Miller, Douglass Turner, "A Progressive Multi-Pass Method for Global Illumination", *Computer Graphics (SIGGRAPH '91 Proceedings)*, vol. 25, no. 4, pp. 165-174 August 1991.

[Chen91b] Chen, Shenchang Eric, "Implementing Progressive Radiosity with User Provided Polygon Display Routines", in *Graphics Gems II*, James Arvo ed., Boston, 1991.

[ChenH90] Chen, Hong, En-Hua Wu, "An Efficient Radiosity Solution for Bump Texture Generation", *Computer Graphics (SIGGRAPH '90 Proceedings)*, vol. 24, no. 4, pp. 125-134, August, 1990.

[Cohen85] Cohen, Michael, Donald P. Greenberg, "The Hemi-Cube: A Radiosity Solution for Complex Environments", *Computer Graphics (SIGGRAPH '85 Proceedings)*, vol. 19, no. 3, pp. 31-40, Aug. 1985.

[Cohen86] Cohen, Michael, Donald P. Greenberg, Dave S. Immel, Philip J. Brock, "An Efficient Radiosity Approach for Realistic Image Synthesis", *IEEE Computer Graphics and Applications*, vol. 6, no. 3, pp. 26-35, March 1986.

[Cohen88] Cohen, Michael, Shenchang Eric Chen, John R. Wallace, Donald P. Greenberg, "A Progressive Refinement Approach to Fast Radiosity Image Generation", *Computer Graphics (SIGGRAPH '88 Proceedings)*, vol. 22, no. 4, pp. 75-84, Aug. 1988.

[Cook82] Cook, Robert L., Kenneth E. Torrance, "A Reflectance Model for Computer Graphics", *ACM Transactions on Graphics*, vol. 1, no. 1, pp. 7-24, 1982.

[Cook84] Cook, Robert L., Thomas Porter, Loren Carpenter, "Distributed Ray Tracing", *Computer Graphics (SIGGRAPH '84 Proceedings)*, vol. 18, no. 3, pp. 137-145, July 1984.

[Cook86a] Cook, Robert L., "Stochastic Sampling in Computer Graphics", *ACM Transactions on Graphics*, vol. 5, no. 1, pp. 51-72, Jan. 1986.

[Cook86b] Cook, Robert L., "Practical Aspects of Distributed Ray Tracing", *SIGGRAPH '86 Developments in Ray Tracing seminar notes*, Aug. 1986.

[Crow84] Crow, F.C., "Summed-Area Tables for Texture Mapping", *Computer Graphics (SIGGRAPH '84 Proceedings)*, vol. 18, no. 3, pp. 207-212, July 1984.

[Dorsey91] Dorsey, Julie, Francois Sillion, Donald Greenberg, "Design and Simulation of Opera Lighting and Projection Effects", *Computer Graphics (SIGGRAPH '91 Proceedings*, vol. 25, no. 4, pp. 41-50, August 1991.

[Frey87] Frey William H., "Selective Refinement: A New Strategy for Automatic Node Placement in Graded Triangular Meshes", *International Journal of Numerical Methods in Engineering*, vol. 24, pp. 2183-2200, 1987.

[Foley90] Foley J. D., A. Van Dam, et. al., *Computer Graphics, Principles and Practice* Addison-Wesley Publishing Co., 1990.

[Fujimoto86] Fujimoto, A., T. Tanaka, K. Iwata, "ARTS : Accelerated Ray Tracing", *IEEE Computer Graphics and Applications*, vol. 6, no. 4, pp. 16-26, 1986.

[George90] David W. George, A Francois X. Sillion Donald P. Greenberg, "Radiosity Redistribution for Dynamic Environments", *IEEE Computer Graphics and Applications* vol. 10, no. 4, pp. 26-34, July 1990.

[Glasner90] Glasner, Andrew S., et. al., *Graphics Gems*, Academic Press, Boston, 1990.

[Goldsmith87] Goldsmith, J., J. Salmon, "Automatic Creation of Object Hierarchies for Ray Tracing", *IEEE Computer Graphics and Applications*, vol. 7(5), pp. 14-20, May 1987.

[Goral84] Goral, Cindy M., Kenneth E. Torrance, Donald P. Greenberg, Bennett Battaile, "Modeling the Interaction of Light Between Diffuse Surfaces", *Computer Graphics (SIGGRAPH '84 Proceedings)*, vol. 18, no. 3, pp. 212-22, July 1984.

[Gour71] Gouraud, Henri, "Continuous Shading of Curved Surfaces", *IEEE Transactions on Computers*, June, 1971.

[Greenberg86] Greenberg, Donald P., Michael Cohen, Kenneth E. Torrance, "Radiosity: A Method for Computing Global Illumination", *Visual Computer*, vol. 2, pp. 291-297, 1986.

[Greenberg90] Greenberg, Donald P., Michael Cohen, Roy Hall, "Radiosity", *SIGGRAPH '90 Course Notes*, July, 1990.

[Greenberg91] Greenberg, Donald P., Michael Cohen, Roy Hall, Holly Rushmeier, Francois Sillion, John Wallace, "Radiosity", *SIGGRAPH '91 Course Notes 11*, July, 1991.

[Haines91a] Haines, Eric, "Ronchamp: A Case Study for Radiosity" *SIGGRAPH '91 Course Notes : Frontiers in Rendering*, pp. 4-1,4-5, July, 1991.

[Haines91b] Haines, Eric, USENET article, Oct 8, 1991.

[Haines92a] Haines, Eric, John R. Wallace "Shaft Culling for Efficient Ray-Cast Radiosity", 3D/Eye Inc., New York, January, 1992.

[Haines92b] Haines, Eric, "Beams O Light: Confession of a Hacker", 3D/Eye Inc., New York, January, 1992.

[Haeb90] Haeberli, Paul, Kurt Akeley, "The Accumulation Buffer: Hardware Support for High-Quality Rendering", *Computer Graphics (SIGGRAPH '90 Proceedings)*, vol. 24, no. 4, pp. 120-125 August, 1990.

[Hall83] Hall, Roy A., Donald P. Greenberg, "A Testbed for Realistic Image Synthesis", *IEEE CGA*, vol. 3, no. 8, pp. 10-20, November, 1983.

[Hall89] Hall, Roy, *Illumination and Color in Computer Generated Imagery*, Springer-Verlag, New York, 1989.

[Hanrahan90] Hanrahan90, Pat., David Salzman, "A Rapid Hierarchical Radiosity Algorithm for Unoccluded Environments", *Technical Report CS-TR-281-90*, Princeton, University, 1991.

[Hanrahan91] Hanrahan, Pat, David Salzman, Larry Aupperle, "A Rapid Hierarchical Radiosity Algorithm", *Computer Graphics (SIGGRAPH '91 Proceedings)*, vol. 25, no. 4, pp. 197-206, August 1991.

[He91] He, Xiao D., Kenneth E. Torrance, Francois Sillion, Donald Greenberg, "A Global Illumination Solution for General Reflectance Distributions", *Computer Graphics (SIGGRAPH '91 Proceedings)*, vol. 24, no. 4, pp. 175-186, August, 1991.

[Hecht79] Hecht, Eugene, Alfred Zajac, *Optics*, Addison-Wesley, Massachusetts, 1979.

[Heckbert84] Heckbert, Paul S., Pat Hanrahan, "Beam Tracing Polygonal Objects", *Computer Graphics (SIGGRAPH '84 Proceedings)*, vol. 18, no. 3, pp. 119-128, July 1984.

[Heckbert90] Heckbert, Paul S., "Adaptive Radiosity Textures for Bidirectional Ray Tracing", *Computer Graphics (SIGGRAPH '90 Proceedings)*, vol. 24, no. 4, pp. 145-154, August, 1990.

[Sarofim67] Sarofim, Adel F., Hottel, Hoyt C., "Radiative Transfer", McGraw Hill, New York, 1967.

[IES81] Illuminating Engineering Society of North America, *IES Lighting Handbook Reference Volume*, Illuminating Engineering Society of North America, 1981.

[Immel86] Dave S. Immel, Cohen, Michael, Donald P. Greenberg, "A Radiosity Method for Non-Diffuse Environments", *Computer Graphics (SIGGRAPH '86 Proceedings)*, vol. 20, no. 4, pp. 133-142, Aug. 1986.

[Kajiya86] Kajiya, James T., "The Rendering Equation", *Computer Graphics (SIGGRAPH '86 Proceedings)*, vol. 20, no. 4, pp. 143-150, Aug. 1986.

[Kolb] Craig Kolb, Rod Bogart, *Raytrace*, Yale University.

[Malley88] Malley, Thomas J. V., "A Shading Method for Computer Generated Images", *Master's Thesis, University of Utah*, June, 1988.

[Mantyla88] Mantyla, Martti, *An Introduction to Solid Modeling*, Computer Science Press, Rockville, Maryland, 1988.

[Maxwell86] Maxwell, Gregory M., Michael J. Bailey, Victor W. Goldschmidt, "Calculations of the Radiation Configuration Factor Using Ray Casting", *Computer-Aided Design*, vol. 18, no. 7, pp. 371-379, Sept. 1986.

[Meyer86] Meyer, Gary W, Holly E. Rushmeier, Michael F. Cohen, Donald P. Greenberg, Kenneth E. Torrance, "An Experimental Evaluation of Computer Graphics Imagery", *ACM Transactions on Graphics*, vol. 5, no. 1 pp. 30-50, Jan. 1986.

[Meyer–Arendt89] Meyer–Arendt, Jurgen R., *Introduction to Classical and Modern Optics*, Prentice-Hall, London, vol. Third Edition, 1989.

[Newell72] Martin E. Newell, R. G. Newell, T. L. Sancha, "A Solution to the Hidden Surface Problem", *Proceedings of the ACM National Conference*, pp. 443-450, 1972.

[Nishita85] Nishita, Tomoyuki, Eihachiro Nakamae, "Continuous Tone Representation of Three-Dimensional Objects Taking Account of Shadows and Interreflection", *Computer Graphics (SIGGRAPH '85 Proceedings)*, vol. 19, no. 3, pp. 23-30, July 1985.

[Nishita86] Nishita, Tomoyuki, Eihachiro Nakamae, "Continuous Tone Representation of Three-Dimensional Objects Illuminated by Sky Light", *Computer Graphics (SIGGRAPH '86 Proceedings)*, vol. 20, no. 4, pp. 125-132, Aug. 1986.

[Nusselt78] Nusselt, Wilhelm, "Graphische Bestimmung der Winkelverhaltwisses bei der Warmestrahlang", *VDIZ*, vol. 72, pp. 673, 1978.

[Phong75] Phong, Bui-Tuong, "Illumination for Computer-Generated Pictures", *Communications of the ACM* vol. 18, no. 3, pp. 311-317, 1975.

[Pueyo91] Pueyo, Xavier, "Diffuse interreflections. Techniques for form-factor computation: a survey", *The Visual Computer*, vol. 7, no 4., pp. 200-209, 1991.

[Recker90] Recker, Rodney J., David W. George, Donald P. Greenberg, "Acceleration Techniques for Progressive Refinement Radiosity", *Computer Graphics (1990 Symposium on Interactive 3D Graphics)*, vol. 24, no. 2, pp. 59-66, March 1990.

[Rost86] Rost, Randi J., *OFF (Object File Format)* model for geometric modeling, Workstation Systems Engineering, Digital Equipment Corporation, November, 1986.

[Rubenstein81] Rubenstein, R. Y., *Simulation and the Monte Carlo Method*, John Wiley and Sons, New York, 1981.

[Rush86] Rushmeier, Holly E., "Extending the Radiosity Method to Transmitting and Specularly Reflecting Surfaces", Masters Thesis, Department of Mechanical Engineering, Cornell University, 1986.

[Rush87] Rushmeier, Holly E., "The Zonal Method for Calculating Light Intensities in the Presence of a Participating Medium", *Computer Graphics (SIGGRAPH '87 Proceedings)*, vol. 21, no. 4, pp. 293-302, July 1987.

[Rush90a] Rushmeier, Holly E., Kenneth E. Torrance, "Extending the Radiosity Method to Include Specularly Reflecting and Translucent Materials", *ACM Transactions on Graphics*, vol. 9, no. 1, pp. 1-27, Jan. 1990.

[Rush90b] Rushmeier, Holly E., Daniel R. Baum, David E. Hall, "Accelerating the Hemi-Cube Algorithm for Calculating Radiation Form Factors", *5th AIAA/ASME Thermophysics and Heat Transfer Conference*, Seattle, Washington, June 1990.

[Rush90c] Rushmeier, Holly E., Stephen D. Tynor, "Incorporating the BRDF into an Infrared Scene Generation System", *Conference on Characterization, Propagation and Simulation of Infrared Scenes, SPIE Proceedings*, Orlando, Florida, vol. 1311, April 1990.

[Samet90] Samet, Hanan, *Applications of Spatial Data Structures (Computer Graphics, Image Processing, and GIS)*, Addison-Wesley, New York, 1990.

[Segal88] Segal, Mark, Carlo H. Sequin, "Partitioning Polyhedral Objects into Nonintersecting Parts", *IEEE Computer Graphics and Applications*, vol. 8, no. 1. January, 1988.

[Shao88] Shao, Min-Zhi, Qun-Sheng Peng, You-Dong Liang, "A New Radiosity Approach by Procedural Refinements for Realistic Image Synthesis", *Computer Graphics (SIGGRAPH '88 Proceedings)*, vol. 22, no. 4, pp. 93-101, Aug. 1988.

[Shinya89] Shinya, Mikio, Takafumi Saito, Tokiichiro Takahashi, "Rendering Techniques for Transparent Objects", *Proceedings of Graphics Interface '89*, pp. 173-181, June, 1990.

[Shirley90] Shirley, Peter, "A Ray Tracing Method for Illumination Calculation in Diffuse-Specular Scenes", *Proceedings of Graphics Interface '90*, pp. 205-212, May, 1990.

[Siegel81] Siegel, Robert, John R. Howell, "Thermal Radiation Heat Transfer", Hemisphere Publishing Corporation, New York, 1981.

[Siegel82] Siegel, Robert, John R. Howell, "A Catalog of Radiation Configuration Factors", McGraw Hill, New York, 1982.

[Sillion89] Sillion, Francois, Claude Puech, "A General Two-Pass Method Integrating Specular and Diffuse Reflection", *Computer Graphics (SIGGRAPH '89 Proceedings)*, vol. 23, no. 3, pp. 335-344, 1989, July.

[Sillion90] Francois Sillion, Claude Puech, Christophe Vedel, "Improving Interaction with Radiosity-based Lighting Simulation Programs", *Computer Graphics (1990 Symposium on Interactive 3D Graphics)*, vol. 24, no. 2, pp. 51-57, March 1990.

[Sillion91] Francois Sillion, James R. Arvo, Stephan Westin, Donald Greenberg, "A Global Illumination Solution for General Reflectance Distributions" *Computer Graphics (SIGGRAPH '91 Proceedings)*, vol. 25, no. 4, pp. 187-196, August 1991.

[Sparrow63] Sparrow, E.M., "A New and Simpler Formulation for Radiative Angle Factors", *Transactions of the ASME, Journal of Heat Transfer*, vol. 85, no. 2, pp. 81-88, 1963.

[Sparrow78] Sparrow, E.M., R.D. Cess, "Radiation Heat Transfer", Hemisphere Publishing Corporation, Washington, 1978.

[Thibault87] Thibault W., Naylor B., "Set Operations on Polyhedron Using Binary Space Partitioning Tress", *Computer Graphics (SIGGRAPH '87 Proceedings)*, vol. 21, no. 4, pp. 153-162, July, 1987.

[Varga62] Varga, Richard, "Matrix Iterative Analysis", Prentice-Hall, New Jersey, 1962.

[Von Herzen87] Von Herzen, Brian, Alan J. Barr, "Accurate Triangulations of Deformed Intersecting Surfaces", *Computer Graphics (SIGGRAPH '87 Proceedings)*, vol. 21, no. 4, pp. 103-110, July, 1987.

[Wallace87] Wallace, John R., Michael F. Cohen, Donald P. Greenberg, "A Two-Pass Solution to the Rendering Equation: A Synthesis of Ray Tracing and Radiosity Methods", *Computer Graphics (SIGGRAPH '87 Proceedings)*, vol. 21, no. 4, pp. 311-320, July 1987.

[Wallace89] Wallace, John R., Kells A. Elmquist, Eric A. Haines, "A Ray Tracing Algorithm for Progressive Radiosity", *Computer Graphics (SIGGRAPH '89 Proceedings)*, vol. 23, no. 3, pp. 315-324, July 1989.

[Wang90] Wang, Yigong, Wayne A. Davis, "Octant Priority for Radiosity Image Rendering", *Proceedings of Graphics Interface '90*, pp. 83-91, May 1990.

[Ward88] Ward, Gregory J., Francis M. Rubinstein, Robert D. Clear, "A Ray Tracing Solution for Diffuse Interreflection", *Computer Graphics (SIGGRAPH '88 Proceedings)*, vol. 22, no. 4, pp. 85-92, Aug. 1988.

[Watt90] Watt, Mark, "Light-Water Interaction using Backward Beam Tracing", *Computer Graphics (SIGGRAPH '90 Proceedings)*, vol. 24, no. 4, pp. 377-385, August 1990.

[Weghorst84] Weghorst, Hank, Donald P. Greenberg, Gary Hooper, "Improved Computational Methods for Ray Tracing", *ACM Transactions on Graphics*, vol. 3, no. 1, pp. 52-68, January 1984.

[Whitted80a] Whitted, Turner, "An Improved Illumination Model for Shaded Display", *Communication of the ACM*, vol. 23, no. 6, pp. 343-349, June 1980.

[Whitted80a] Whitted, Turner, S. Rubin, "A Three-dimensional Representation for Fast Rendering of Complex Scenes", *Computer Graphics (SIGGRAPH '80 Proceedings)*, vol. 14, no. 3, pp. 110-113, July, 1980.

[Wolberg90] Wolberg, George, *Digital Image Warping*, IEEE Computer Society Press Monograph, Las Alamitos, California, 1990.

[Wyvill86] Wyvill, Geoff, L. Tosiyasu, "Space Subdivision for Ray Tracing in CSG", *IEEE Computer Graphics and Applications*, vol. 6, no. 4, pp. 28-34, April 1986.

[SIG91] *SIGGRAPH '91 Tutorial (11) on Radiosity*, Las Vagas, August 1991.