B.Tech. Project

Efficient Radiosity-based Illumination Solutions

Submitted in partial fulfillment of the requirements for the degree of
Bachelor of Technology

by
Chetan Rai
92005036

under the guidance of
Dr. Sharat Chandran

Department of Computer Science and Engineering
Indian Institute of Technology
Bombay

1995-96

Abstract

This report approaches the goal of photorealism by following, in the main, a radiosity approach. The basic radiosity method is described along with several enhancements including the progressive refinement approach. The problem of patch generation is considered and some algorithms which extend radiosity to non-diffuse environments are discussed. The report proposes a new form factor algorithm for radiosity solutions. The new algorithm substantially reduces aliasing problems inherent in the hemicube method without giving up several of the advantages of the hemicube. The algorithm has its basis in the technique of area anti-aliasing in image rendering. Our method preprocesses the input surfaces, and there is also an extra computational cost involved in scan conversion of patches as well as occlusion testing; however, this expense is less than the extra cost associated with an increase in hemicube resolution to improve image quality. Further, the method can be embedded in several existing radiosity algorithms such as adaptive subdivision and progressive refinement radiosity. Finally, the extended area anti-aliasing can also be used in applications other than radiosity (such as image rendering of complex scenes on a raster display device).

Contents

List of Figures

Chapter 1

Introduction

Realistic image synthesis has always been a major goal in computer graphics. For this purpose, a variety of illumination models have been developed. Various models consider one or more phenomena which arise from interactions between light and the surfaces in a scene, such as reflection, transmission, absorption and refraction. Though realism depends upon a correct modeling of all of these phenomena, reflection models in computer graphics have generally simulated only certain aspects of these and only to the extent that is computationally possible within the constraints of the equipment available. For example, the radiosity method (which is what we are primarily concerned with in this report) in its basic form limits itself to handling only interactions between diffusely reflecting surfaces. We start with the principles of radiative transfer since a physically correct lighting model must be based on these principles.

1.1 Principles of Radiative Transfer

1.1.1 The Nature of Light

Light is electro-magnetic energy and can be expressed as an energy flux density, q, in units of energy per unit time per unit area. When q represents the total energy flux density leaving a surface, it is referred to as the radiosity B of the surface. To account for the directional character of light energy, it is necessary to introduce the radiant intensity I. With angles as defined in Figure 1.1, the intensity leaving a surface element $\mathrm{d}A$ in a direction (θ, ϕ) is given by

$$I(\theta, \phi) = \frac{\mathrm{d}q(\theta, \phi)}{\cos\theta \mathrm{d}\omega} \tag{1.1}$$

where $\mathrm{d}q(\theta, \phi)$ is the differential energy flux density leaving the surface within a small solid angle $\mathrm{d}\omega$ in the given direction. Intensity is found by expressing $\mathrm{d}q$ per unit *projected* area (thus introducing $\cos\theta$) per unit solid angle (dividing by $\mathrm{d}\omega$).

For synthetic image generation, intensity is the appropriate quantity with which to work. The energy flux density reaching the eye from a surface in a scene varies with

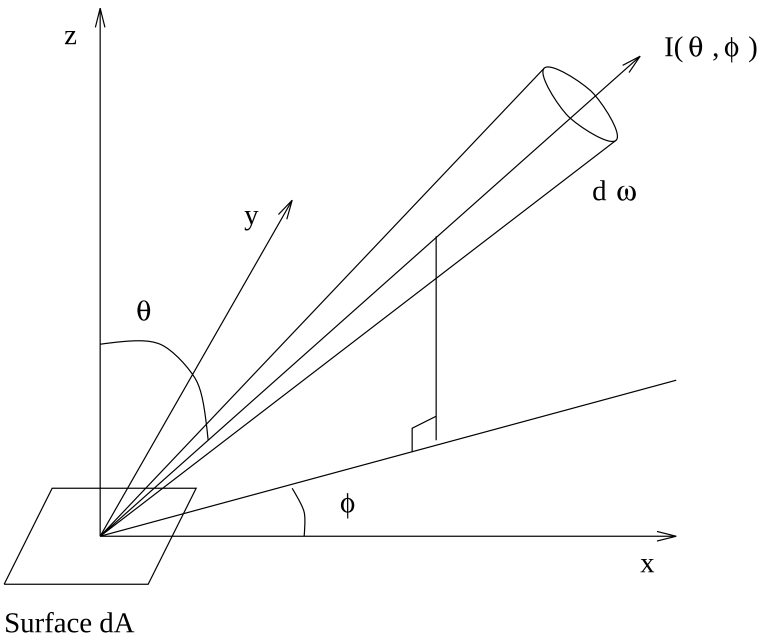


Figure 1.1: Geometry of light of intensity I leaving a surface element dA

distance according to an inverse square law. Similarly, the solid angle subtended by a surface as viewed by the eye also varies according to an inverse square law. The intensity, or brightness, perceived by the eye is proportional to the ratio of these two quantities. Thus, the intensity of a surface in a scene remains constant to an observer as the observer moves toward or away from a scene.

1.1.2 Interaction of Light with Surfaces

Light is assumed to be emitted, reflected, absorbed or transmitted only at surfaces within an environment. To construct a synthetic image of a scene, it is necessary to know the intensity of light leaving a surface in the scene. This intensity, $I_o(\theta_o, \phi_o)$, is composed of emitted, reflected and transmitted light:

$$I_o(\theta_o, \phi_o) = I_e(\theta_o, \phi_o) + I_r(\theta_o, \phi_o) + I_t(\theta_o, \phi_o) \tag{1.2}$$

The second and third terms in Equation 1.2 couple the intensity leaving the surface to the intensities arriving at it from all of the other surfaces in the environment. The incident light is related to the reflected and transmitted light by two distributions called the *bidirectional reflectance* and *bidirectional transmittance*. These are defined as ratios of the reflected or transmitted intensities to the incident energy:

$$\rho_{bd}(\theta_r, \phi_r, \theta_i, \phi_i) = \frac{\mathrm{d}I_r(\theta_r, \phi_r)}{\mathrm{d}q_i(\theta_i, \phi_i)} \tag{1.3}$$

$$\tau_{bd}(\theta_r, \phi_r, \theta_i, \phi_i) = \frac{\mathrm{d}I_t(\theta_r, \phi_r)}{\mathrm{d}q_i(\theta_i, \phi_i)} \tag{1.4}$$

Here, the subscript i refers to the direction of incident light.

Using the definition of intensity from Equation 1.1 and integrating Equation 1.3 over the hemisphere of incident directions, the intensity of light reflected from the surface is given by

$$I_r(\theta_o, \phi_o) = \int_\Omega \rho_{bd}(\theta_r, \phi_r, \theta_i, \phi_i) I_{incident}(\theta_i, \phi_i) \cos\theta_i \mathrm{d}\omega_i \qquad (1.5)$$

We can similarly get an expression for I_t (the hemisphere used for integration is the *back* of the surface). In a closed environment, this incident intensity is the outgoing intensity of another surface and we get a system of integral equations, the solution of which will give us complete information about the intensity distribution across all the surfaces in the scene.

1.1.3 The Nature of Surfaces

The bidirectional reflectance and transmittance are determined by the nature of the surface. Based on how the light incident on a surface is reflected back or transmitted into the environment, surfaces are broadly classified into *diffuse* or *specular*. We will restrict ourselves in this section to reflecting surfaces - transmitting surfaces have analogous properties.

Specular Surfaces

Specular surfaces are those in which the incident light is wholly reflected in the mirror direction and the intensity of reflected light in any other direction is zero. Reflections from such surfaces follow the laws of reflection of light and are called specular reflections. Examples of specular surfaces are mirrors or other polished surfaces.

$$\rho_{bd}(\theta_r, \phi_r, \theta_i, \phi_i) = \begin{cases} \rho_s & \text{if } (\theta_r, \phi_r) \text{ is the mirror direction for } (\theta_i, \phi_i), \\ 0 & \text{otherwise.} \end{cases} \qquad (1.6)$$

Ray tracing is an ideal method for determining the interactions between light and specular surfaces. The shadows cast by specular interactions in the environment are sharp.

Diffuse Surfaces

Diffuse surfaces are those in which the incident light is reflected with equal intensity in all directions. Radiosity methods are very effective in determining the diffuse interactions between surfaces. Using Equation 1.1, the total energy leaving a diffuse surface in all directions per unit time per unit area is

$$\int_\Omega \mathrm{d}q(\theta, \phi) = \int_\Omega I \cos\theta \mathrm{d}\omega \qquad (1.7)$$

$$= \pi I \qquad (1.8)$$

where the integration is done over the hemisphere in front of the surface.

Practical Surfaces

Though there are surfaces with reflectance distributions very different from either diffuse or specular (e.g., surfaces which polarize light), real life environments generally contain surfaces which are a combination of the two. In such a case, the reflected intensity is *spread* in all outgoing directions with a *high* component in the mirror direction.

1.2 Overview of the Report

Chapter 2 introduces the radiosity method and Chapter 3 discusses some major advances and refinements of the basic algorithm. Chapter 4 looks at extensions of the radiosity method to include environments with non-diffuse surfaces. Chapter 5 describes a new radiosity algorithm with anti-aliasing, and is followed by the concluding remarks.

Chapter 2

The Radiosity Method

Many environments consist primarily of surfaces which reflect light diffusely. In such environments, direct illumination and object-to-object reflection between diffuse surfaces may account for the overwhelming proportion of the light energy present. In order to generate images which realistically simulate an actual scene, the physical behavior of visible light, as it is propagated through the environment must be modeled. Since the intensity and the distribution of light in a scene are governed by the energy transport and conservation principle, these must be considered for accurately simulating different light sources (primary, as well as secondary) in the same scene.

First introduced to computer graphics in [Goral et al., 1984] for simple environments and extended to more complex non-convex environments in [Cohen and Greenberg, 1985], the radiosity method is based on energy principles and may be applied monochromatically or to finite wavelength intervals. In its simplest form, the key assumption is that all surfaces are ideal diffuse (Lambertian) reflectors and emitters. The procedure is applicable to arbitrary environments composed of such surfaces and it can account for direct illumination from a variety of light sources and all multiple reflections within the environment. Since small specular areas may contribute little to the total light energy, such small specular interactions may be later added to the diffuse interaction solutions, with minimal error. Effects such as color bleeding, which are very common in diffuse interactions, can be precisely modeled by the radiosity method.

A major advantage of the radiosity method is that the resultant surface intensities are independent of the viewer's position. The environmental intensity information can be preprocessed for viewing dynamic sequences of the given scene from different angles. This makes the method suitable for applications in architectural and interior designing.

2.1 Radiosity Basics

The radiosity method describes an equilibrium energy balance within an *enclosure*. An enclosure is a set of surfaces that completely defines the illuminating environment. The walls of the enclosure consist of light sources and reflecting walls, and one or more of the surfaces may be fictitious (e.g., an open window). Our first step is to divide the environment into a set of small polygonal areas called *patches*. Once we have *discretized* the environment, we will concern ourselves only with these patches and not the surfaces of which they form a part. Since the basic radiosity method considers the intensity across a patch to be constant, the accuracy of our solution will depend on this discretization.

The *radiosity* of a patch is the energy it emits per unit time per unit area. Since the outgoing energy is the sum of the energy it emits and the incident energy it reflects, we have the following relation for a patch i:

$$B_i \mathrm{d}A_i \quad = \quad E_i \mathrm{d}A_i + \rho_i \int_j B_j \mathrm{d}A_j F_{ji} \tag{2.1}$$

If we consider a discretization into n finite areas instead of the infinitesimal areas in Equation 2.1, we get

$$B_i A_i \quad = \quad E_i A_i + \rho_i \sum_{j=1}^{n} B_j A_j F_{ji} \tag{2.2}$$

Here, A_i is the area of patch i, E_i is the energy it emits, and F_{ji} is the fraction of energy leaving patch j which arrives at patch i. This fraction is generally called the *form factor* from patch j to patch i. Using the reciprocity relationship for form factors $F_{ij}A_i = F_{ji}A_j$, which we will derive in Section 2.2, and dividing Equation 2.2 by A_i, we obtain the more commonly used radiosity equation.

$$B_i \quad = \quad E_i + \rho_i \sum_{j=1}^{n} B_j F_{ij} \tag{2.3}$$

Writing the system of equations obtained by considering the above relation for all patches, we get

$$\begin{bmatrix} 1-\rho_1 F_{11} & -\rho_1 F_{12} & \cdots & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & 1-\rho_2 F_{22} & \cdots & -\rho_2 F_{2n} \\ \vdots & \vdots & & \vdots \\ -\rho_n F_{n1} & -\rho_1 F_{n2} & \cdots & 1-\rho_n F_{nn} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{bmatrix} \tag{2.4}$$

The nxn matrix in Equation 2.4 is called the radiosity matrix. Gauss-Seidel iteration is used to obtain a solution to this system of equations, with the initial guess being simply the emission of each patch.

To render the image the discretized radiosity information is used to create a continuous shading across a given surface. A scheme of bilinear interpolation can be used within each patch. In order to perform the interpolation, radiosity values must be transferred from the patches themselves to each vertex of the patches. For each surface, the radiosities of patch vertices which are interior to the polygon containing the patch are computed as the average of radiosities of the surrounding patches. Exterior vertex radiosities are extrapolated values from the adjacent interior vertex radiosities. Finally, the resulting vertex radiosities are

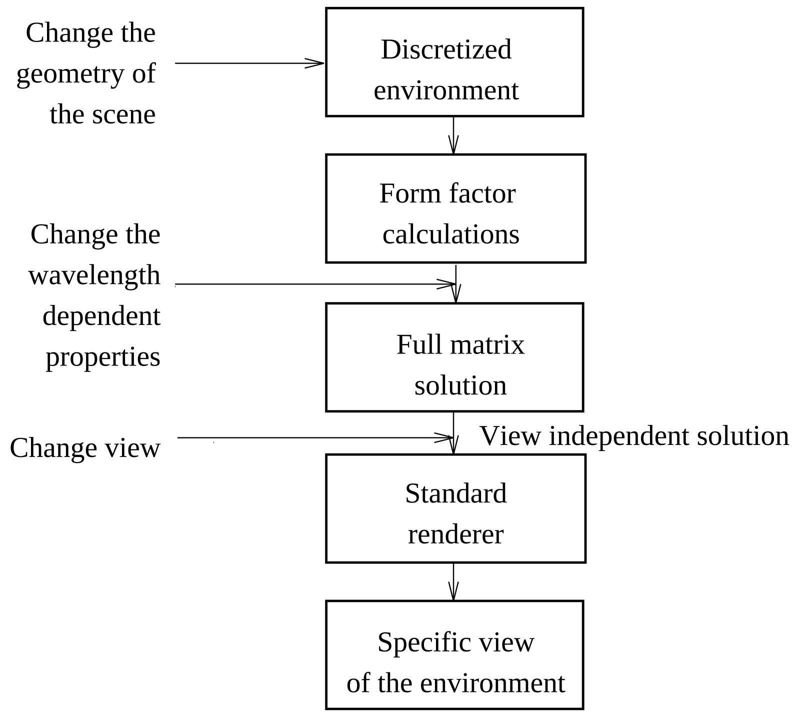


Figure 2.1: Program flow for the full matrix radiosity method

used for bilinear interpolation within each patch. This radiosity solution is independent of view position and direction. Standard techniques can now be used for display.

The program flow model for this radiosity solution, also called the *full matrix solution*, is given in Figure 2.1.

2.2 Form Factors

The computation of form factor values in the radiosity method is a distinctly non-trivial high-cost operation (except for cases where both the shape and the relationships of patches are highly constrained). Besides the time required, the storage cost (using the standard full matrix formulation described in Section 2.1) is $O(n^2)$. This is huge even if many form factors are zero and the matrix is highly sparse. It is therefore not surprising that a lot of research has focussed on making form factor computation more efficient, both in terms of time and space.

2.2.1 Formulation

As shown in Figure 2.2, consider the form factor between two infinitesimal surfaces with differential areas $\mathrm{d}A_i$ and $\mathrm{d}A_j$. The solid angle subtended by $\mathrm{d}A_j$ at $\mathrm{d}A_i$ is given by:

$$\mathrm{d}\omega_{ij} = \cos\phi_j \frac{\mathrm{d}A_j}{r^2}$$

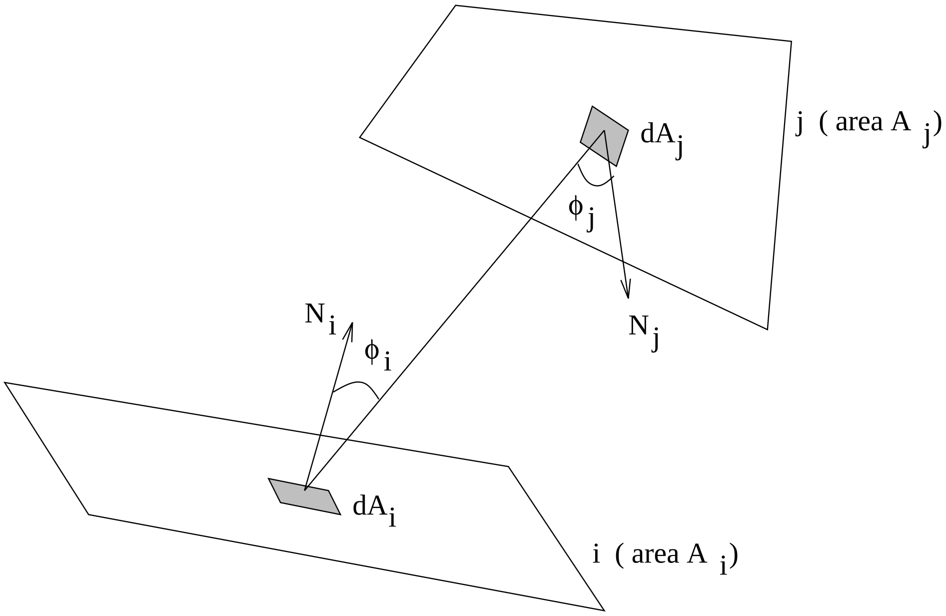


Figure 2.2: Form factor geometry of two patches A_i and A_j

If I_i is the outgoing intensity, the energy leaving the differential area $\mathrm{d}A_i$ that reaches $\mathrm{d}A_j$ per unit time is given by:

$$\begin{aligned}\mathrm{d}E_i\mathrm{d}A_i &= I_i \cos\phi_i \mathrm{d}\omega_{ij}\mathrm{d}A_i \\ &= I_i \cos\phi_i \cos\phi_j \mathrm{d}A_i \frac{\mathrm{d}A_j}{r^2} \qquad (2.5)\end{aligned}$$

Note that $\mathrm{d}E_i$ is the energy leaving $\mathrm{d}A_i$, per unit time, per unit area (unprojected), through a solid angle $\mathrm{d}\omega$ subtended by the patch $\mathrm{d}A_j$.

The total energy leaving the elemental area $\mathrm{d}A_i$ in all directions into the hemisphere centered on that area is $E_i\mathrm{d}A_i$. From the definition of form factors,

$$F_{dA_i dA_j} = \frac{\text{Radiative energy reaching } \mathrm{d}A_j \text{ from } \mathrm{d}A_i}{\text{Total energy leaving } \mathrm{d}A_i \text{ in all directions}}$$

Using Equations 1.8 and 2.5, we get:

$$\begin{aligned}F_{dA_i dA_j} &= \frac{I_i \cos\phi_i \cos\phi_j \mathrm{d}A_i \mathrm{d}A_j}{\pi I_i \mathrm{d}A_i r^2} \\ &= \frac{\cos\phi_i \cos\phi_j \mathrm{d}A_j}{\pi r^2} \qquad (2.6)\end{aligned}$$

The above equation represents the differential-to-differential area form factors. The differential-to-finite area form factors can be obtained by integrating Equation 2.6 over patch j:

$$F_{dA_i j} = \int_{A_j} \frac{\cos\phi_i \cos\phi_j}{\pi r^2} \mathrm{d}A_j \qquad (2.7)$$

The patch-to-patch form factor is obtained by taking the area average over patch i, since the intensity I_i is independent of the direction and the position over patch i:

$$F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{\cos\phi_i \cos\phi_j}{\pi r^2} \mathrm{d}A_j \mathrm{d}A_i \tag{2.8}$$

In order to consider the possible occlusion of the surface j from surface i, we define a function called $H(i,j)$ as follows:

$$H(i,j) = \begin{cases} 1 & \text{if patch } i \text{ can see patch } j, \\ 0 & \text{otherwise} \end{cases}$$

We include this function in our formula for form factors to take care of occlusion:

$$F_{ij} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \frac{\cos\phi_i \cos\phi_j}{\pi r^2} H(i,j) \mathrm{d}A_j \mathrm{d}A_i \tag{2.9}$$

The very form of the above equation imposes a severe computational constraint. Calculating the form factors between two elemental areas is straightforward. Evaluation of the differential-to-finite area form factors requires one area integral, whereas calculating the finite-to-finite area form factors requires two area integrals. Such integrals are very expensive to evaluate analytically for real life geometries.

Before we go on to actually calculating form factors, let's look at a few simply properties which can be obtained directly from the above formulation. Firstly, from Equation 2.9, we directly get the reciprocity relationship $F_{ij}A_i = F_{ji}A_j$ used in Section 2.1. Secondly, in a closed environment, the energy leaving any patch i must reach *some* other patch, which implies $\sum_{j=1}^{n} F_{ij} = 1$. We therefore have the property that the radiosity matrix is diagonally dominant. This means that Gauss-Seidel iteration is guaranteed to converge in a small number of iterations (Section 2.1). Finally, since flat or convex surfaces cannot see themselves, we have $F_{ii} = 0$ for such surfaces.

2.2.2 The Hemicube Method

For evaluating the double area integral of Equation 2.9, we can use Stokes' Theorem to convert it to a double contour integral, but evaluation of this is also computationally expensive and is only appropriate for simple environments. An occluding or intervening patch between patches i and j changes the shape of the contours and the determination of such a shape would be extremely difficult if the number of intervening patches is large. Numerical techniques can provide a more efficient means to compute form factors for complex environments. Starting from a geometric analog to the analytic derivation, a numerical method was proposed in [Cohen and Greenberg, 1985] to approximate the form factors.

Consider Equation 2.9. If the distance r between the two patches is large compared to their size, the value of the inner integral does not change much over the range of the outer integral (provided there are no intervening patches). Then, the effect of the outer integral is simply a multiplication by 1 and finding a solution to the inner integral will provide a good approximation. If the patches are close together relative to their size, or there is partial occlusion, the patches can be subdivided into smaller patches and the single integral approximation can still be used.

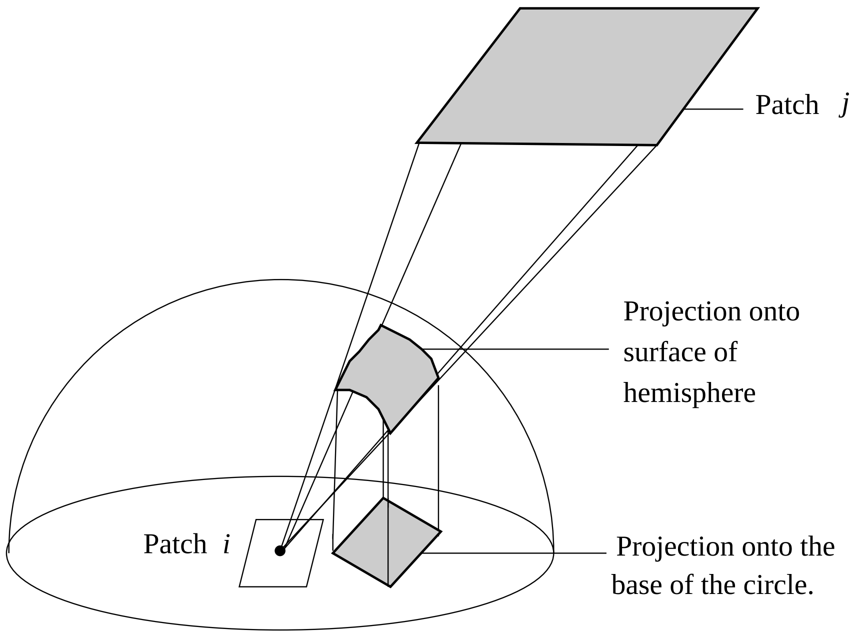


Figure 2.3: The Nusselt analog

A geometric analog for this form factor integral was developed by Nusselt and has been used to obtain form factors (Figure 2.3). For a finite area, the form factor is equivalent to the fraction of the circle (which is the base of the hemisphere) covered by projecting the area onto the hemisphere and then orthographically down onto the circle. The projection onto the hemisphere accounts for the $1/r^2$ term as well as the cosine of the angle between the normal of the projecting patch and the r vector. The orthographic projection onto the circle has the effect of multiplying by the cosine of the angle between the r vector and the normal of the receiving patch. The π in the denominator accounts for the area of a circle with unit radius.

This hemisphere can be broken into small delta solid angles, each representing a *delta form factor*. The delta form factor is equal to the delta solid angle multiplied by the cosine of the angle from the normal. The form factor to a patch equals the sum of the delta form factors covered when projecting the patch onto the hemisphere. If all the patches in the environment are projected onto the hemisphere (removing more distant patches in case of overlap), the final projection and summations would provide the form factors to all patches from the patch represented at the center of the hemisphere. This procedure would then intrinsically include the effects of hidden surfaces.

In order to perform the summation of delta form factors, we must discretize the hemisphere. An evaluation can then be made as to which patch projects onto that discrete area. Difficulty in creating equal sized elements on a sphere as well as creating a set of linear co-ordinates to uniquely describe locations on its surface makes this approach impractical.

From the definition of the form factor it can be seen that any two patches in the environment, which when projected onto the hemisphere occupy the same area and location, will have the same form factor value. Instead of projecting onto a hemisphere, therefore, we construct an imaginary cube around the center of the patch and project onto the *hemicube* above the "horizon". We then have five faces which can be divided into square "pixels" and the environment can be clipped to the frusta using a standard polygon clipper. A modified z-buffer algorithm will take care of the case when two patches project onto the same pixel of the hemicube - instead of maintaining intensity information, an item buffer is maintained of which patch is seen at each pixel on the hemicube.

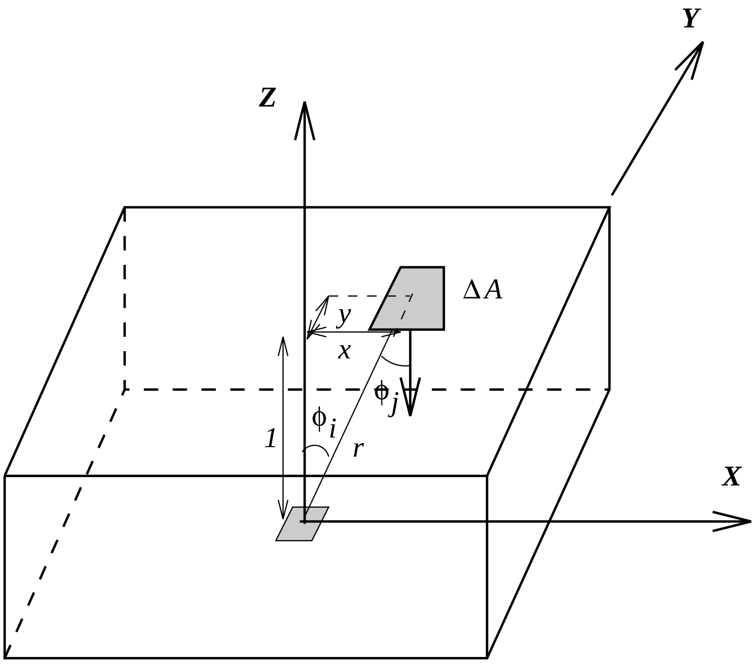


Figure 2.4: Geometry of delta form factors for a pixel on the top of the hemicube

The delta form factors can be easily determined from the geometry. From Figure 2.4, we get

$$\Delta\text{Form Factor} = \frac{1}{\pi(x^2 + y^2 + 1)^2}\Delta A$$

for the top of the hemicube and, similarly,

$$\Delta\text{Form Factor} = \frac{z}{\pi(y^2 + z^2 + 1)^2}\Delta A$$

for the sides. These values are stored in a lookup table for greater efficiency (this table need only contain values for one-eighth of the top face and one quarter of one side face due to symmetry).

2.2.3 Problems with the Hemicube Method

The hemicube is an extremely efficient method for determining form factor values but the assumptions it makes can lead to erroneous results.

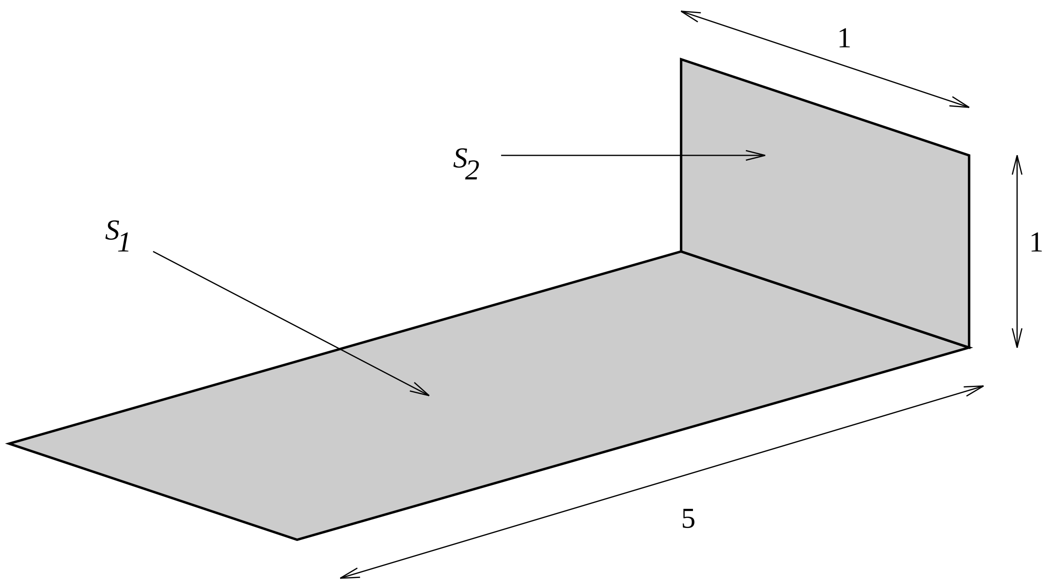


Figure 2.5: Two perpendicular surfaces of dissimilar size

The assumption of proximity (the distance between surfaces is much greater than their size) is violated when two surfaces are adjacent to each other as in Figure 2.5. The hemicube method makes a relatively good approximation of F_{21} since the area of patch 2 is much smaller than the area of patch 1 but a poor estimate is made of F_{12}.

The assumption of visibility requires that the function H (Section 2.1) stay constant across a patch for any hemicube pixel. The single point evaluation of H is a possible source of significant error (see Figure 2.6).

There is a further assumption that the patches project onto a whole number of pixels. This finite resolution of pixels, as elsewhere in computer graphics, can lead to aliasing errors. Though these intensity errors are not visually detectable in most cases, significant errors and disturbing visual artifacts are produced when calculating form factors to or from light sources. Though aliasing can be reduced by increasing the hemicube resolution, this also increases the computational cost by a significant amount. In Chapter 5, we shall look at a new method to reduce aliasing without a substantial cost increase.

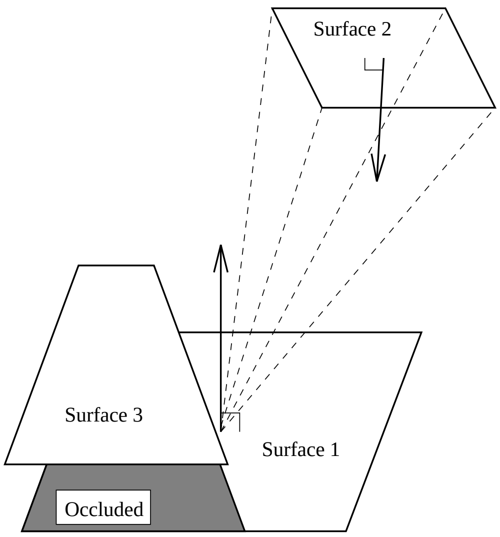


Figure 2.6: Geometry where the estimate of F_{12} using the hemicube is poor, because of an intervening patch 3

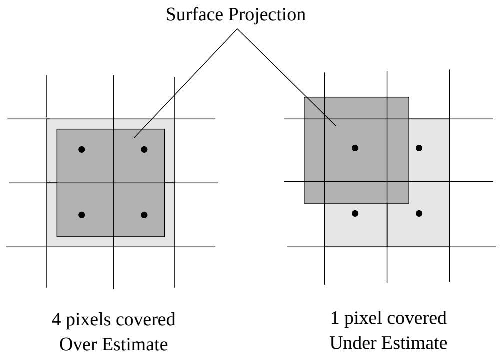


Figure 2.7: Hemicube aliasing can cause over/under estimation of the projected surface area.

Chapter 3

Advances in Radiosity

We have described the basic radiosity algorithm in Chapter 2. The calculation of form factors and the solution of the radiosity matrix takes an enormous amount of time for any significant number of patches. Also, the storage cost is $O(n^2)$ if n is the number of patches. We look at a technique which takes up only $O(n)$ space and also gives fast approximate solutions which improve with time. An algorithm which gives greater accuracy without the use of a correspondingly larger number of patches and a technique for generating a "good" set of patches from the set of initial surfaces are also discussed. We end this chapter with a fast hierarchical radiosity algorithm which recursively subdivides patches though the order of the number of interactions considered is lower than the square of the number of subdivided patches.

3.1 The Progressive Refinement Approach

A progressive refinement approach to radiosity was proposed by [Cohen et al., 1988]. This is the radiosity algorithm in a restructured form and differs from the full matrix solution primarily in two aspects:

- The radiosities of all patches are updated simultaneously.
- The patches are processed in a sorted order according to their energy contribution to the environment.

To further improve the quality of the images generated during the earliest stages of the algorithm, an estimate of the global illumination is obtained directly from the known geometrical and reflective characteristics of the environment. This estimate is gradually replaced by more exact information as the solution progresses, providing a graceful and continuous convergence to a realistic image.

3.1.1 Shooting versus Gathering

In the full matrix algorithm, the Gauss-Seidel method converges to a solution by solving the system of equations one row at a time. The evaluation of the ith row of the equation provides an estimate of the radiosity of patch i based on the current estimate of the radiosities of all other patches. In a sense, the light leaving patch i is determined by *gathering* in the light from the rest of the environment.

It can be seen from Equation 2.3 that the contribution of the radiosity of patch i to the radiosity of patch j is

$$\begin{aligned} B_j \text{ due to } B_i &= \rho_j B_i F_{ji} \\ &= \rho_j B_i F_{ij} A_i / A_j \end{aligned} \tag{3.1}$$

Thus the total contribution to the environment from the radiosity of patch i is given by the above equation by going over all patches j. The point to be noted is that while this equation adds radiosity to patches j, the form factors used, F_{ij}, are still the form factors calculated using the hemicube placed at patch i. Thus, each step of the solution now consists of performing a single hemicube over a patch and adding the contribution from the radiosity of the patch to the radiosities of all other patches. This is, in a sense, *shooting* light out from the patch into the environment.

During the course of the iterative solution, this step may be repeated for patch i several times as the solution converges. Each time the estimate of the radiosity the patch i will be more accurate. Since the environment will already include the contribution from the previous estimates of B_i, only the difference ΔB_i between the previous and current estimates of B_i needs to be considered at each step. ΔB_i represents the *unshot radiosity*. The solution step may be restated as shown in the algorithm in Figure 3.1. All radiosities B_i and ΔB_i are initialized to E_i.

```
for each iteration, for each patch i {
   calculate the form factors F(i,j) using a hemicube at patch i;
   for each patch j {
     ΔRad = ρ(j) ΔB(i) F(i,j) A(i)/A(j);
     ΔB(j) = ΔB(j) + ΔRad; (* update change since the last time
       patch j shot light *)
     B(j) = B(j) + ΔRad; (* update total radiosity of patch j *)
   }
   ΔB(i) = 0; (* reset unshot radiosity for patch i to zero *)
}
```

Figure 3.1: Algorithm for shooting the increments in radiosity

The above procedure continues until the solution converges to within the desired threshold. Convergence is guaranteed because the radiosity "vector" is bounded above (each component is bounded by $\frac{\sum_{i=1}^{n} E_i}{1-\rho_{\max}}$ where $\rho_{\max}$ is the maximum of all ρ_i's) and is monotonically increasing with each iteration (B_i's are always updated by positive quantities; at least one B_i is updated in each iteration otherwise the solution has already converged).

Each intermediate step simultaneously improves the solution for many patches, providing intermediate results which can be displayed as the algorithm proceeds.

3.1.2 Solving in a Sorted Order

In addition to converging gracefully, it is desirable that the solution improves in accuracy as quickly as possible. The final radiosity B_j of a given patch j consists of the sum of the contributions from all other patches. The final value of this sum will be approached earliest in the process if the largest contributions are added first. These will tend to come from those patches which radiate the most energy, i.e., which have the largest product $B_i A_i$.

The algorithm is implemented by always shooting from the patch for which the product $\Delta B_i A_i$ is maximum. Most light sources are automatically processed first by this rule, since initially, we set $\Delta B_i = E_i$ for all patches. Since light sources are typically the most significant source of illumination for many patches, following the initial processing of light, much of the environment will already be well illuminated. The next set of patches processed according to this rule will be those that received the most light from the light sources, and so on.

When solving in a sorted order, the solution tends to proceed in approximately the same order as the light would propagate through the environment. This generally provides an accurate solution in less than a single iteration, thus substantially reducing the computational costs.

3.1.3 The Ambient Term

If we use the procedure mentioned above, intermediate images will progress from a dark environment, continuously brightening to a fully illuminated scene including all diffuse interreflections. The illumination of the scene in the early stages will be inadequate, particularly for the regions which do not receive direct illumination, since global illumination is not yet accurately represented. Therefore, we add an ambient term, only for display purposes, and this is **not** taken into account by the solution itself. The contribution of the ambient term gracefully decreases as the solution continues, providing a useful image almost immediately which unobtrusively progresses to an accurate rendering.

Computation of the Ambient term

We start by approximating the form factor from any patch to a patch j as the fraction of the total area of the environment that is occupied by the patch j. As with correct form factors, this sums to unity:

$$F_{*j} \approx \frac{A_j}{\sum_{j=1}^{n} A_j}$$

The average reflectivity of the environment is computed as the area weighted average of the patch reflectivities:

$$\rho_{\text{ave}} = \frac{\sum_{i=1}^{n} \rho_i A_i}{\sum_{i=1}^{n} A_i}$$

In order to take the effect of multiple reflections into account, we define the overall interreflection factor R as the geometric sum:

$$R = 1 + \rho_{\text{ave}} + \rho_{\text{ave}}^2 + \rho_{\text{ave}}^3 + \cdots = \frac{1}{1 - \rho_{\text{ave}}}$$

Using these definitions, the *ambient radiosity* is calculated by the formula:

$$\text{Ambient} = R \sum_{j=1}^{n} (\Delta B_j F_{*j})$$

Thus, at any point in the computation, the estimate of the radiosity of each patch can be improved by adding the contribution of the ambient term. The improved estimate of the radiosities is given by:

$$B_i' = B_i + \rho_i \text{Ambient}$$

Since this ambient term is not added to ΔB_i, it is not shot during the solution. It is used only for display purposes. As the solution progresses, the unshot energy (and therefore the ambient term) decreases. When the solution is complete, B_i and B_i' have the same value.

3.1.4 Ray Tracing to Compute Form Factors

The limitations of the hemicube method have already been discussed. [Wallace et al., 1989] present a new method for computing form factors - ray tracing is used to perform the numerical integration of the form factor equation (Equation 2.8). This method is particularly useful in the context of a progressive radiosity approach, where form factors are calculated with respect to only one source at each step of the solution. The use of ray tracing ensures adequate sampling by allowing illumination to be computed directly at the element vertices. By tracing rays from the vertices to uniformly distributed points representing finite delta areas on the source, smooth shading can be achieved for a very small number of rays.

The differential-to-finite area form factor $F_{dA_j i}$ is given by Equation 2.7. Numerical integration may be accomplished in a straightforward manner by approximating the integral as the sum of form factors computed for smaller regions of area ΔA_i:

$$F_{dA_j i} \quad = \quad \sum_{k=1}^{n} \frac{\cos\theta_{ik} \cos\theta_{jk}}{\pi r_k^2} \Delta A_i \tag{3.2}$$

However, Equation 3.2 breaks down if ΔA_i is large relative to the distance; as r becomes less than one, ΔA_i must shrink correspondingly, or the result grows without bound.

To limit the subdivision of the source and thus the cost of computing form factors, the delta areas are treated explicitly as finite areas. An equation for the form factor from finite delta areas can be obtained by approximating each delta area by a simple finite geometry (a geometry for which an analytical solution is available). Analytical solutions for a variety of configurations can be found. Because of its simplicity, a disk has been chosen as the finite area upon which to base the approximation. The form factor from a differential area $\mathrm{d}A_j$ to a *directly opposing* disk of area A_i and radius a is given by

$$F_{dA_j i} = a^2/(r^2 + a^2) \tag{3.3}$$

Using the reciprocity principle, the form factor from a disk source to a differential area receiver can be obtained:

$$F_{idA_j} = \mathrm{d}A_j/(\pi r^2 + A_i) \tag{3.4}$$

This relationship can be generalized by making some approximations. The effect of different orientations of the source and receiving areas can be approximated be including the cosines of the angles between the normal at each surface and the direction between the source and the receiver.

$$F_{idA_j} = \mathrm{d}A_j \cos\theta_i \cos\theta_j/(\pi r^2 + A_i) \tag{3.5}$$

(Here, θ_i and θ_j retain their earlier meanings.) A further approximation is to assume that the area is disk shaped. The accuracy of this approximation depends, of course, on the area's actual shape.

To evaluate the form factor from a general source to a vertex, the source is divided into delta areas. The form factor due to each delta area is computed using Equation 3.5. Occlusion is tested by shooting a single ray from the receiving vertex to the center of the delta area on the source. If sample points are distributed uniformly on the source then the total form factor is simply the sum of the form factors computed for each sample point k:

$$F_{idA_j} = \mathrm{d}A_j \frac{1}{m} \sum_{k=1}^{m} H(j, i_k) \frac{\cos\theta_{i_k} \cos\theta_{j_k}}{\pi r_k^2 + A_i/m} \tag{3.6}$$

where m is the number of sample points on the source and H is the occlusion function of Section 2.2.

Equation 3.6 has a number of desirable characteristics. The presence of the area in the denominator prevents the result from growing without bound for small r. This allows approximate form factors to be computed using large delta areas, and hence, a low number of sample points. Thus, a fast, approximate radiosity solution can be obtained using a very small number of rays (as few as one ray per vertex per source!). The accuracy of the result increases as the number of sample points is increased. This can be seen by observing that as m grows larger the term A_i/m tends towards zero and each term of the summation approaches the equation for a differential-to-differential area. Thus, a smooth continuum of results is available depending on the amount of work one is willing to do.

By substituting Equation 3.6 in Equation 3.1, we get the radiosity at *vertex* j due to patch i:

$$B_j \text{ due to } B_i = \rho_j B_i A_i \frac{1}{m} \sum_{k=1}^{m} H(j, i_k) \frac{\cos\theta_{i_k} \cos\theta_{j_k}}{\pi r_k^2 + A_i/m} \tag{3.7}$$

At each step of the progressive solution the contribution made by the current source to the radiosity of each element vertex in the environment is computed using Equation 3.7.

3.2 Substructuring and Adaptive Subdivision

To improve image quality the regions in the environment that exhibit high intensity gradients must be subdivided. The additional refinements should not substantially affect computation time, and should be performed automatically without user intervention. [Cohen et al., 1986] describe enhancements to the radiosity method that satisfy these requirements.

Using the method described in Section 2.1, we obtain a coarse patch solution (i.e. a solution in which the initial discretization of the environment is not very fine). After having computed the vertex radiosities, we determine if for any patch, the variation between the radiosities at its vertices is more than a certain threshold. If this is the case, the patch is further subdivided into still finer patches, which are referred to as *elements*. Now, we assume that the radiosity over the patch may vary, but the radiosity over each element of the patch must remain constant. Since each element has a unique geometric position relative to all other patches, the element-to-patch form factors will vary across the patch. This process of discretizing a patch into elements is called *substructuring*.

Let us assume that a patch i has been substructured into R elements. Using the hemicube method, we determine the form factors from all elements i_q (i_q is the qth element of patch i) to each patch j in the environment. These form factors are represented by $F_{i_q j}$. Note that in this case, the hemicube is placed at the center of the element and the patches in the environment are projected onto the surface of the hemicube. Even if some other patch has been substructured into elements, it is the patch and not the elements that is projected. We are not at all interested in finding the element-to-element form factors.

After having computed all element-to-patch form factors $F_{i_q j}$, we determine the form factors F_{ij} by taking the area average:

$$F_{ij} = \frac{1}{A_i} \sum_{q=1}^{R} F_{i_q j} A_{i_q}$$

These values of F_{ij} are more accurate than the values calculated using the coarse patch discretization and hence, these new values are substituted for F_{ij} in the radiosity matrix. A similar procedure is applied for each patch which has been identified for substructuring. Thus, we again get a radiosity matrix of the same size as that in the coarse patch solution, but this one has more accurate F_{ij} values for all patches i which have been substructured.

The matrix is again solved using the Gauss-Seidel iterations and a new set of values for B_i is obtained. From these patch radiosity values, we obtain the element radiosities using:

$$B_{i_q} = E_{i_q} + \rho_{i_q} \sum_{j=1}^{n} B_j F_{i_q j}$$

From these radiosities, we recompute the radiosities at the vertices of the patches or elements and determine if further substructuring of some elements is required. All such elements are identified and the above process is repeated. This continues until no more substructuring is required.

Since this process of substructuring and recomputation of the radiosities takes place adaptively without any intervention of the user, this approach is known as *adaptive subdivision.*

A closer examination of this procedure shows that the element-to-patch form factors are nothing less than a discretization for numerically integrating the outer integral of Equation 2.8. As a result, this method removes the restriction (in the hemicube method) that patches must be small relative to the distances which separate them. In general, there are far more elements than patches. Thus, for an image of given quality, the form factor calculations are greatly reduced. The element radiosities are finally found from the patch radiosities and the element-to-patch form factors. The result is that, with fewer patches and far smaller computation and storage requirements, a more accurate solution and a superior rendering is obtained.

3.3 Adaptive Mesh Generation

The effectiveness of the method described in Section 3.2, as with most iterative solutions, depends on the initial subdivision of the environment into patches. Small objects casting shadows can easily be missed if the patch size is too large. Further, since emitting surfaces are not subdivided, accurate soft shadowing is not done. Finally, the subdivision of receiving polygons into a regular grid can lead to the phenomenon of "light leaks" where the grid partitions along a surface do not coincide with light-occluding polygons abutting the surface and thus allow light to leak past the occluding polygons (Figure 3.2). [Campbell and Fusell, 1990] present a radiosity approach which automatically generates an initial surface mesh by efficiently subdividing input surfaces along shadow boundaries. This initial mesh is then adaptively refined using both further shadow boundary subdivision and intensity gradient refinement. Radiosity computations are done using a ray tracing approach like that of [Wallace et al., 1989], although ray tracing is not required for visibility calculations due to the shadow boundary subdivision. The presented technique alleviates the problems of potentially missed objects and light leaks.

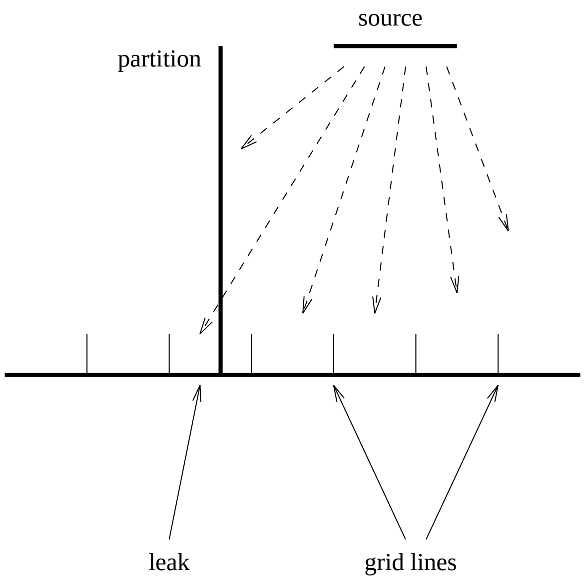


Figure 3.2: Light leaking

3.3.1 The Data Structure

The method described in this section uses the BSP (Binary Space Partitioning) tree structure. Before describing the algorithm, it will be useful to go over the basics of this data structure which has a number of applications in computer graphics, including hidden surface removal, shadow generation and acceleration of ray tracing.

A BSP tree can be used to represent a set of faces in spaces by recursively subdividing the space along hyperplanes determined by the orientations of the polygons within the space. Polygons may be chosen in any order to determine these partitioning hyperplanes, with the most desirable order being one which results in the fewest faces being split along hyperplane boundaries (for some applications, however, the minimization of the height of the tree is more desirable). The resulting data structure is a binary tree, in which each interior node represents a partitioning hyperplane and its defining face, along with any other faces in the same hyperplane that may exist in the input database, and the leaf nodes represent convex volumes of space determined by the partitioning.

As described in [Fuchs et al., 1980], BSP trees can be used to determine the visibility priority of a collection of polygons for any viewing position. This is achieved by using a modified inorder traversal of the tree. Each node is processed recursively by inserting the coordinates of the viewing position into the equation of the partitioning hyperplane at that node. The sign of the result indicates which half of space (defined by the hyperplane) the viewing position is in. If the viewing position is in one of the half-spaces, the subtree representing that half-space is processed first. If the viewing position is on the plane, the subtrees can be processed in any order. Once the subtree has been processed, the polygons within the current node can be output. This is followed by processing the other subtree. Thus the exact order of traversal is determined by the viewing position, and it is guaranteed that this order will output polygons in front to back order relative to the viewing position.

3.3.2 Overview of the Algorithm

The initial mesh for the algorithm is a set of input polygons. In order to facilitate the sorting of polygons and elements involved in shadow determination, as well as to perform visible surface determination when progressive rendering is done, the input polygons are used to generate a BSP tree called the pBSP tree.

For each iteration of the illumination process, we process the patch with most untransmitted energy as the light source. First, the pBSP tree is used to determine the nearest front-facing receiver polygon. If the solid angle subtended by the source at this polygon is small enough, the source can be approximated by a point light source. If it fails the test, we check if it has already been subdivided by previous calculations. If so, the check is applied recursively to elements deeper in the subdivision hierarchy. If bottom level sub-polygons are too large, they are subdivided along their long axis until the size criterion is met.

Visibility is determined through the use of a BSP tree based shadow algorithm. As a source element is being processed, each of the other polygons in the scene is considered in order of increasing distance from the centroid of the source using the BSP tree. The first step in the processing of one of these polygons is to determine whether any of the polygons closer to the source casts a shadow on the polygon under consideration. If so, the polygon is split along the shadow boundary (note that sharp shadows generated using the source centroid as a point light source are used). Once this splitting is complete for all potentially shadowing polygons, the receiving polygon has been subdivided into elements, each of which can be considered totally visible or totally hidden from the source.

Form factors for each of the visible receiving elements are calculated. If these differ by more than a user-specified tolerance, the receiving element is subdivided (along its largest dimension) and form factors recalculated for each resulting element until the tolerance is met.These form factors are normalized to ensure energy balance and are then used to compute radiosities and sample the incoming energy at each receiving element vertex.

After illumination, we add the current polygon to the list of potentially shadowing polygons and move on to the next receiver. When one element has fully scattered its light, the process is repeated for the remaining source elements on the current light source polygon. Once the polygon has distributed all its available energy, we move to the next brightest source polygon and repeat the process. This is continued until the total untransmitted energy has converged to zero.

3.4 Fast Hierarchical Radiosity

[Hanrahan et al., 1991] present a radiosity algorithm which constructs a hierarchical representation for the form factor matrix by adaptively subdividing patches according to an error bound, thus ensuring that all form factor values are calculated to the same precision. The proposed algorithm was inspired by recent developments in the solution of the N-body problem which is similar to form factor calculation in the sense that there are n interacting objects with $O(n^2)$ interactions which fall off as $1/r^2$ if r is the distance between the objects.

3.4.1 Form Factors

The algorithm starts with the set of input polygons as patches. For each pair of patches, the form factor is estimated. If this estimated value is greater than a user-supplied bound (say F_ϵ), the patches are recursively subdivided and the form factors are therefore calculated between smaller patches.

The form factor from a point to a disk (a disk of the same area is used to approximate a patch of possibly different geometry) is bounded by $(a/r)^2$, where a is the radius of the disk and r is the distance from the point to the center of the disk, while the error grows by terms involving powers of a/r if this is used as the form factor estimate (see Section 3.1.4). Since the form factor grows as $(a/r)^2$, the differential form factor is a good estimate of the correct form factor when the size of the patch is small compared to the distance between the patches. This is ensured since patches are subdivided if the estimate is greater than the bound F_ϵ.

[Hanrahan et al., 1991] also show that the number of interactions is $O(n)$, where n is the number of nodes in the hierarchy of patches. Two patches of a size a can interact only if the distance between them is greater than $a/\sqrt{F_\epsilon}$. So, a patch p in the interior of the hierarchy has to handle only the interactions with the "neighbours" (patches which are closer than the threshold) of its parent and will pass on the interactions with its neighbours to its children. This ensures that each patch interacts with a constant number of other patches giving the $O(n)$ bound.

So far there has been no mention of visibility between patches. Occluding surfaces can never increase the light transport between surfaces. Therefore, extra subdivision of patches is never required due to occlusion and the criterion discussed earlier (for subdivision) will work. The effect of occlusion can be modeled, after calculating the form factor without taking occlusion into account, by weighting this form factor by the fraction each patch sees of the other. This fraction is determined by firing jittered rays between the two polygons. The recursive subdivision procedure can use the results of visibility tests. If two patches become totally invisible with respect to each other, no further subdivision is required to determine their interaction to within the error bound. If, on the other hand, two patches become completely visible, no further visibility tests are required for their descendants.

3.4.2 The Radiosity Solution

After calculation of the form factors, the radiosity matrix has to be solved. Jacobi iteration can be used to gather light in the following manner; shooting can be done with a similar procedure. Every patch in the hierarchy first gathers light from the patches it interacts with (there are a constant number of these). The total amount of energy received by a patch in the hierarchy is the energy it gathers and the energy gathered by its ancestors.

Therefore, after gathering, there is a downward pass from the root of the hierarchy to its leaves in which the energy gathered is pushed down to the leaf nodes followed by an upward pass in which the radiosity of each patch is set equal to the area-weighted average of the radiosities of its children. This updates the energies for the next iteration.

A variation, similar to the idea of multigridding, is to refine the hierarchy as the iteration proceeds. The advantage is that a coarse solution can be performed cheaply and provides a better starting point for successive more expensive solutions. In this algorithm, multigridding is equivalent to refining the hierarchy of patches with smaller and smaller F_ϵ's. Further, the refinement can now be based on BF (if B is the radiosity and F the form factor), the amount of energy transported, instead of on F. This results in refinement being put off until there is some energy to be transported, making the early iterations even cheaper.

Chapter 4

Towards a Global Illumination Solution

In chapter 2, we have introduced the radiosity method as a means to finding the illumination solution for an environment consisting of diffusely reflecting and emitting surfaces. However, besides the diffuse-to-diffuse interaction, there are other interactions between surfaces, viz. specular-to-diffuse, diffuse-to-specular, specular-to-specular and others (between surfaces which do not fall into the categories of either diffuse or specular or anywhere in between). Ray tracing deals with specular-to-specular interactions correctly and with specular-to-diffuse interactions empirically. It is to be expected that a combination of these two major post-Phong illumination models will yield useful results, but a simple superposition will obviously not work since it will not consider the specular-to-diffuse transport mechanism. Also, if the "other" interactions are to be modeled, the basic radiosity method will have to be extended. This chapter gives a sampling of research which has been conducted to extend radiosity to provide illumination solutions for more general environments.

4.1 Radiosity for Non-Diffuse Environments

[Immel et al., 1986] introduce a general radiosity method accounting for all inter-reflections of light between diffuse and non-diffuse surfaces in complex environments. A complete view-independent description of the light leaving each surface in each direction is computed, allowing dynamic sequences of images to be rendered with little additional computation per image. Phenomena such as *reflection tracking* (reflections following a moving observer across a specular surface) are produced. A correct solution would involve finding the intensity at all points in the scene and in all directions (refer section 1.1). The radiosity method makes this problem tractable by discretizing the surfaces in the scene to patches and by assuming that the intensity in all directions is the same at any point. This method removes the second assumption and instead, discretizes the hemisphere of incoming and outgoing directions.

4.1.1 Formulation

Instead of radiosities of surfaces, we now deal with directional intensities and equation 2.3 changes to

$$I_i(f) = E_i(f) + \sum_{j=1}^{n} \sum_{d=0}^{D-1} \rho_{bd,i}(f,d) I_j(d^{-1}) \cos\theta_d \omega_d H(i,j,d) \tag{4.1}$$

where

$$\begin{aligned}
D &= \text{number of directions} \\
d &= \text{incoming direction} \\
d^{-1} &= \text{outgoing direction for patch } j \text{ corresponding direction } d \text{ for patch } i \\
f &= \text{outgoing direction} \\
\theta_d &= \text{angle between the normal of surface } i \text{ and direction } d \\
\omega_d &= \text{solid angle covered by } \textit{discrete} \text{ direction } d \\
H(i,j,d) &= \begin{cases} 1 & \text{if patch } i \text{ sees patch } j \text{ in direction } d, \\ 0 & \text{otherwise.} \end{cases}
\end{aligned}$$

At first glance, equation 4.1 appears much more complicated than the earlier radiosity equation since intensities must be found for each patch for each direction. However, certain properties of the matrix of equations can be exploited to make it tractable. Since each patch receives energy from only D directions, the number of *non-zero* matrix co-efficients in the matrix is only nD^2 out of a total of $(nD)^2$ co-efficients (for example, if there are 1000 patches and 1000 directions, the matrix is 99.9% sparse). This fact can be taken advantage of in an implementation.

4.1.2 The Global Cube

The earlier radiosity method replaced the hemisphere of incoming and outgoing directions by the upper half of a cube (refer section 2.2.2) and discretized its surface. The hemicube was oriented such that the center of the top face coincided with the patch normal. Since the orientations were in the patch's local co-ordinate system, there was no single directional relationship between hemicubes, but none was required since each patch was assumed to have a single intensity equal in all directions. Now, however, when the intensities are allowed to vary directionally, it is important to equate the incoming directions of the receiving patch with the reciprocal outgoing direction from the sending patches. A full cube oriented along the *global* axes is therefore maintained. Its advantage lies in the fact that if patch i sees patch j in the (x, y, z) direction, then the reciprocal cell from patch j is simply $(-x, -y, -z)$. The cube is stored as a vector of visible patch numbers. If the faces of the cube are oriented as in FIgure 4.1 and the vector is ordered as $+X$,$+Y$,$+Z$,$-Z$,$-Y$,$-X$ in row major order, the reciprocal global cube locations can be trivially calculated. For

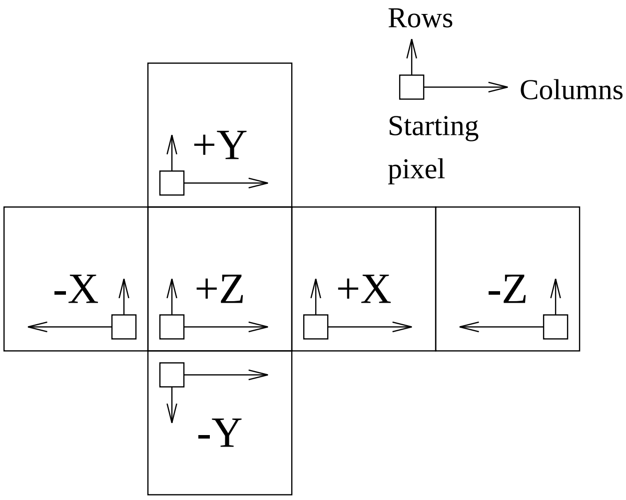


Figure 4.1: The cells within the faces of the global cube are ordered such that the reciprocal global cube locations may be trivially calculated

a global cube with D discrete directions, the cell with the reciprocal direction d^{-1} from a cell with index d will simply have index $D - d - 1$.

The projection of the environment onto the global cube surrounding a vertex provides the H terms in equation 4.1. The θ terms are found by taking the dot product of the patch normal and the orientation of each cell. A solid angle ω is associated with each cell, providing the necessary information to solve for the directional intensities of each patch. A simultaneous solution to the matrix of equations 4.1 is obtained, as earlier, using the Gauss-Seidel iterative technique.

At this point, a view independent radiosity solution has been calculated. The final rendering stage is the same as that in the basic radiosity method except that the intensities pointing in the direction of the viewer have to be extracted from the global cubes.

4.2 A Synthesis of Ray Tracing and Radiosity

In section 4.1, the view-independent radiosity method was extended to include specular reflection. A simultaneous solution of the system of equations obtained gives an intensity in each direction for each patch. Unfortunately, for specular surfaces, the intensity as seen by the viewer typically changes very quickly from pixel to pixel. Therefore, a view-independent solution may require that specular surfaces be subdivided to the point where each patch covers approximately one pixel in the final image. Although this approach would eventually converge to an accurate solution as the discretization of the environment increased, the computation and space required to solve the resulting system of equations precludes subdivision to the required level. As a result, artifacts appear when specular surfaces are rendered, since the correct intensity is known only at relatively widely spaced sample points. A reasonable approach to solving the global illumination problem is to take

```
for each direction d and patch i {
   intensity(i,d) = emiss(i,d);
}
for each iteration, for each patch i {
   for each direction d {
     newint(i,d) = emiss(i,d);
   } for each direction d {
     energy(d) = intensity(cube(i,d,D − d − 1))(D − d − 1) cos θ ω(d);
     for each direction f {
       newint(i,f) = ρ_bd(i,f,d) energy(d);
     }
   }
   for each direction d {
     compare intensity(i,d) with newint(i,d]; (* check for convergence *)
     intensity(i,d) = newint(i,d);
   }
}
```

Figure 4.2: Algorithm for the global cube method

advantage of the strengths of both view-dependent (ray tracing) and view-independent (radiosity) sampling. Based on these observations, [Wallace et al., 1987] present a two-pass approach to solving the illumination problem.

4.2.1 Two-Pass Image Synthesis

The two-pass approach for computing global illumination described in the following sections accounts for all the mechanisms of light transfer mentioned in the beginning of the chapter. In a view-independent stage of the algorithm, called the *preprocess*, the complete global propagation of light is approximated in order to determine the diffuse component of intensity for all surfaces. All transfer mechanisms are included in the preprocess. Thus, specular reflection must be accounted for, but only to the extent required to determine the diffuse component. The second view-dependent stage, called the *postprocess*, then uses the results of the preprocess as the basis for calculating the non-diffuse component to the accuracy desired by the view. The postprocess accounts for the specular-to-specular and diffuse-to-specular transfer mechanisms, in the latter case using the diffuse component calculated during the preprocess as the source. For each pixel in the final image, the resulting specular component is added to the diffuse component, interpolated from the preprocess sample points, providing a complete solution.

4.2.2 The Preprocess

The standard radiosity solution using the hemicube method provides the basis for the preprocess. Since the standard radiosity method accounts only for diffuse reflection, extensions to the algorithm must be made to add diffuse transmission and approximate specular-to-specular reflection and specular transmission insofar as it affects the diffuse component for all surfaces.

Translucency

Ideal diffuse transmission is readily included by placing a hemicube on the back, as well as the front, of the transmitting surface and calculating *backward* form factors in addition to the usual *forward* form factors.

Specular-to-Diffuse Transport

Specular reflection or transmission may also be accounted for by performing additional work during the calculation of form factors. Specular surfaces are treated as additional routes by which light energy may reach one diffuse surface from others. Thus, two diffuse surfaces that "see" each other through intermediate specular surfaces have an extra form factor. The additional form factors may be calculated by any method that can determine the path of specular reflection or transmission. The approach used in [Wallace et al., 1987] restricts specular surfaces to perfect planar mirrors or filters. This allows the use of the earlier hemicube method by creating a "virtual world" which is seen through the mirror or filter (Figure 4.3).

Result of the Preprocess

The final result of the preprocess is an accurate determination of the diffuse intensity at selected sample points (the element vertices) within the environment. Although some of these points may end up being hidden from a particular viewpoint, the number of points required is typically much less that the number that would have been imposed by a pixel-to-pixel calculation. The hemicube algorithm determines the incoming intensities at each sample point for a number of directions determined by the hemicube resolution. In a sense, the effort that would have been required to determine the diffuse component at every pixel in a view-dependent approach is applied instead to more accurately determining the illumination at a smaller number of points.

4.2.3 The Postprocess

The postprocess takes the results of the preprocess and, for a given view, completes the solution of the illumination equation for the surface visible at each pixel. The view-dependence of the postprocess permits the efficient calculation of the non-diffuse component by limiting the direction for which it must be calculated to the view direction. The non-diffuse

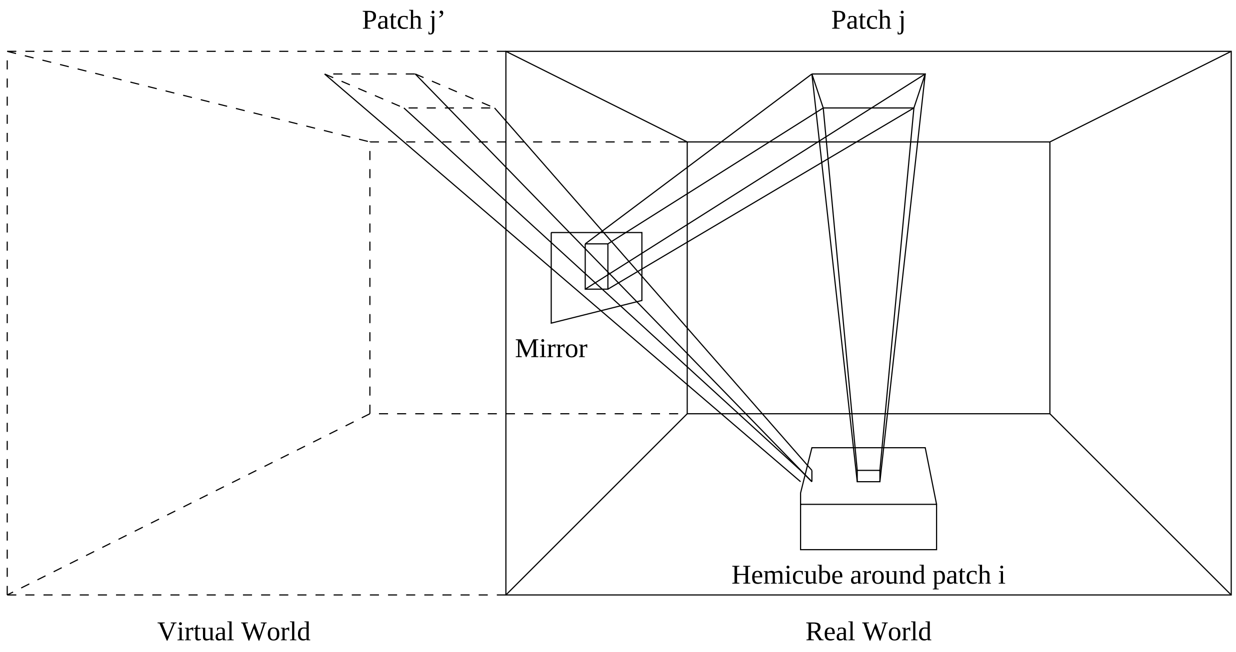


Figure 4.3: Calculation of extra form factors to account for mirror reflection. Patch i receives light directly from patch j and indirectly from patch j'

component depends on the intensities arriving at the surface from the entire hemisphere of directions, weighted by the bidirectional reflectance. Distributed ray tracing and "cone tracing" were the first algorithms for sampling these incoming intensities. The method used by [Wallace et al., 1987] is based on the observation that in most non-diffuse reflection, the bidirectional reflectance is such that only a fraction of the incoming intensities over the hemisphere make a significant contribution to the outgoing intensity in a particular direction (the reflection is then specular-*like*). Thus, with little loss of accuracy, the limits of integration may be reduced from the entire hemisphere to the smaller solid angle of interest.

The Reflection Frustum

A view frustum encompassing this solid angle is constructed. The visible surfaces for this *reflection frustum* are determined using a standard low-resolution z-buffer algorithm (typically of the order of 10x10 pixels). The incoming intensity seen through each pixel of this frustum is simply the intensity of the surface visible at that pixel and may contain both diffuse and specular components. The diffuse component is determined during the z-buffer operation by bilinear interpolation from the vertex intensities calculated during the preprocess. Where a surface visible in a reflection frustum pixel has a specular component, its intensity is calculated by applying the entire postprocess algorithm recursively, as in traditional ray tracing.

Chapter 5

A New Form Factor Algorithm

The radiosity method is a powerful technique for increased realism in computer generated images. The hemicube method was introduced as an efficient numerical method to compute approximate form factors in radiosity. However, as discussed in 2.2.3, it is not without problems.

Aliasing problems occur due to the fact that the resolution of any practical hemicube is finite. The resulting artifacts can be particularly severe in case of form factor calculation to or from light sources. Numerous techniques (e.g., [Baum et al., 1989]) for form factor computation have been proposed which do not have these problems. However, these methods are computationally more expensive than the hemicube method (and radiosity with the hemicube is already very expensive!). Clearly, the hemicube method is worth preserving for its efficiency if we can eliminate or at least reduce these problems without incurring much additional computational expense. This is especially relevant in computer graphics today; increasingly sophisticated and complex techniques have been proposed to exploit hardware innovations on the one hand, while at the same time, there is a tendency to use lower end platforms for advanced graphics applications.

Until now, the only way to reduce aliasing artifacts while retaining the hemicube method was to increase the resolution of the hemicube, which can be likened to supersampling in the context of rendering. In this chapter, we present a modified hemicube algorithm which reduces aliasing by removing the restriction that a patch must project onto a whole number of pixels and allows pixels to be *partially* covered by patches. The method has its basis in simple area anti-aliasing which is described in the next section, along with a polygon scan conversion algorithm which performs the extra function of calculating the area of each pixel covered by the polygon. We then describe our radiosity algorithm in Section 5.2, followed by a discussion of our implementation and the results obtained.

5.1 Simple Area Anti-aliasing

Simple scan conversion algorithms consider each pixel as a mathematical point. The conversion is done by sampling at these "points," and aliasing is the result of undersampling. For example, a solid area scan conversion algorithm might base the determination of whether or not a pixel is inside or outside the area on the location of the center of the pixel. For simple bilevel displays, this technique is necessary, and results in the characteristic stair step or jagged edge at the border. For multilevel displays, the appearance of the solid area or line (in case of line rasterization) can be improved by "blurring;" a simple approach is to let the intensity of a pixel be proportional to the area of the pixel inside the solid area or under the line.

5.1.1 Theoretical Basis

Simple area anti-aliasing is equivalent to pre-filtering with a box function. Though other filters (such as triangular or Gaussian filters) yield better results for anti-aliasing, they are much more complex and the implementations will be computationally expensive when compared to the box filter.

5.1.2 The Solid Area Scan Conversion Algorithm

The solid area scan conversion algorithm for area anti-aliasing needs to do extra work to determine the area of each pixel within the solid area. We could use an ordered edge list algorithm for polygons, working with a pixel traversal algorithm for the edges instead of a line segment drawing algorithm such as the Bresenham algorithm. Also, this pixel traversal algorithm would have to calculate the area under the edge it was traversing; for the interior of the polygon, we know that the entire pixel is covered by the solid area. A different scan conversion algorithm meant especially for area scan conversion is described below. We have used this algorithm in our implementation.

The algorithm desribed in Figure 5.1 uses two arrays, the area array A and the "carry" array C (see Figure 5.2), which hold values for each pixel and are initialized to zero. In the first (edge traversal) phase, we perform a pixel traversal for each edge of the polygon. At each pixel (i, j) crossed by an edge, we add to $A(i, j)$ the area under the part of the edge which lies inside pixel (i, j). $C(i, j)$ is updated by the area which would be covered in a pixel below pixel (i, j) if the part of the edge inside pixel (i, j) were projected downward onto the bottom of the view rectangle (this equals $(x_e - x_l)p_y$, where x_e and x_l are the x co-ordinates of the points of entry and leaving of the edge in the pixel and p_y is the height of a pixel). The areas here are signed areas, negative if the outward normal of the edge points toward the bottom of the view rectangle and positive otherwise. In the second (carry adjustment) phase, we add $\sum_{k=1}^{i-1} C(k, j)$ to $A(i, j)$. This method corresponds to calculating the area of a polygon by calculating signed areas under each edge and summing them; the only difference is that we need the area *in each pixel*.

Note that, for convex polygons, this algorithm has a running time of $O(n_e + n_p)$, if n_e is the number of edges in the polygon and n_p is the number of pixels, which is the same as that for the ordered edge list algorithm which does not find areas (though the constants are larger). In case of general polygons, we have a running time of $O(n_e n_p)$ compared to $O(n_e \log n_e + n_p)$ for the ordered edge list algorithm. This algorithm can be modified easily to work for normal scan conversion, with A and C holding only integer values.

```
initialize arrays A and C to zero
for each edge e of the polygon (* in any order *) {
   get the sign s of the edge e
   for each pixel (i,j) crossed by edge e {
     A(i,j) = A(i,j) + s (x_e - x_l) (y_e + y_l)/2
     (* all co-ordinates are measured from the lower left corner
     of pixel (i,j) *)
     C(i,j) = C(i,j) + s (x_e - x_l) p_y
   }
}
for each column j {
   CC = 0 (* cumulative carry for the column *)
   for each row i from top to bottom {
     A(i,j) = A(i,j) + CC
     CC = CC + C(i,j)
   }
}
(* the array A now holds the area of the polygon within each pixel *)
```

Figure 5.1: Algorithm for area scan conversion

5.2 The Radiosity Algorithm

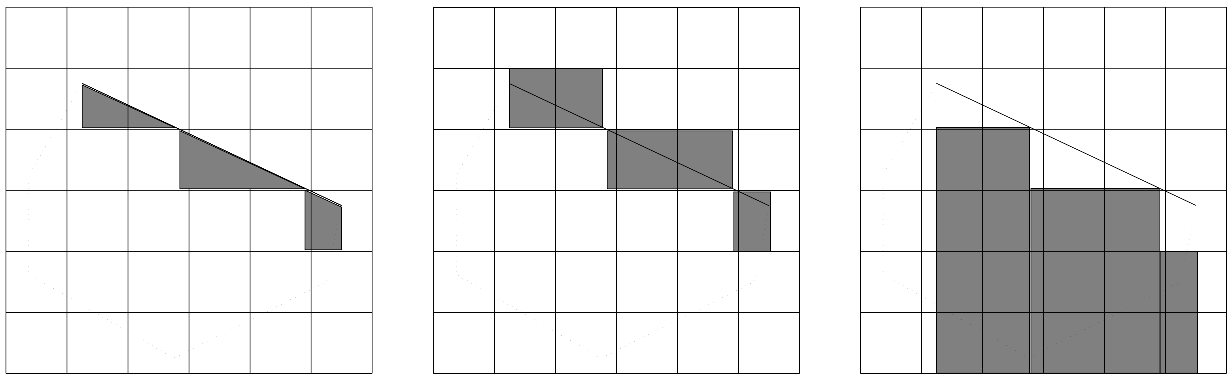
The following sections describe the proposed radiosity algorithm. We note that the extension of simple area anti-aliasing to complex environments described below can also be adapted to anti-aliasing in case of rendering of complex environments.

5.2.1 The Preprocess

Before we generate patches from the set of input polygons, we incur a one-time additional expense (as compared to the original hemicube method) in the construction of a Binary Space Partitioning (BSP) tree of this set of polygons. BSP trees are not new to radiosity: [Campbell and Fusell, 1990] (see Section 3.3) use this data structure to facilitate sorting of patches and elements involved in shadow determination. Note that the use of the BSP tree structure limits the environment to flat surfaces only. Any curved surfaces must be approximated by flat surfaces before the BSP tree is constructed.

5.2.2 Patch Generation

Our algorithm places no constraints on patch generation except that the patches generated must be polygonal, so the input to this part of the radiosity solution must be processed to



(a) (b) (c)

Figure 5.2: These figures show (a) the area by which A is updated, (b) the area by which C is updated and (c) the effect of C for one edge of a polygon

remove curved boundaries. Patch generation may be automatic or performed manually.

5.2.3 Form Factor Determination

As with the original hemicube method, we construct an imaginary unit hemicube around the center of each patch when calculating the form factors from that patch to the other patches in the environment. However, instead of using the z-buffer algorithm to take care of occlusion of patches in the environment, we use the BSP tree and process the patches in front-to-back depth order with respect to the center of the hemicube. Also, instead of a patch number at each hemicube pixel (referring to the closest patch covering that pixel processed so far), we store an estimate e_f of the fraction of the area of the pixel covered by the patches processed so far. The patch in the environment currently being considered is scan converted using the algorithm discussed in Section 5.1.2. Now we have the area the patch covers for each hemicube pixel.

To take into account the effects of occlusion by intervening patches, we estimate the *additional* area (Section 5.2.3) covered by the patch under consideration for each pixel using the fraction e_f. The form factor to the patch is approximated by the sum of the delta form factors of the pixels it covers, each weighted by the (estimate of the) additional fractional area calculated earlier. Before we go on to the next patch in the environment, we update e_f by adding the estimate of the additional fraction of the pixel covered by the current patch to it.

Estimation of Additional Area Covered

The additional area of a pixel covered by the patch is estimated by its expected value. We already know the area covered by the patch without considering occlusion, say A_0, and the fraction e_f. The probability that an infinitesimal area will not be occluded by patches already processed given that it projects onto the pixel is given by the fraction of the area of the pixel not covered by those patches. We use the estimate $(1 - e_f)$ for this value. This leads to the following expression:

$$\begin{aligned} A_E &= \int_{\text{patch}} (1 - e_f)\mathrm{d}A \\ &= A_0(1 - e_f) \end{aligned} \tag{5.1}$$

where A_E is the expected additional area covered, which is the quantity we require.

When the patch has been processed, A_E/A_p is added to e_f (A_p is the area of each pixel of the hemicube).

Normalization

A problem with our method is that all the light energy of the patch around which the hemicube has been constructed might not be distributed to the environment. As an example, consider the situation in Figure 5.3, where two patches p_1 and p_2, in order, each cover half of a pixel. However, the expectation is that half of the second patch will overlap with

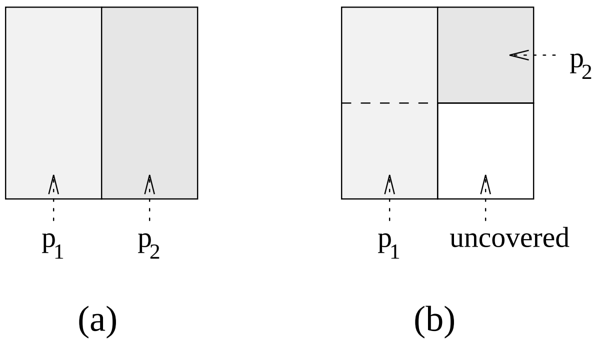


Figure 5.3: (a) Actual situation (b) Expected situation

the first, which implies that a quarter of the delta form factor for that pixel will not be assigned to any patch. To take care of this situation, we add the remaining fraction of the delta form factor for each pixel to the last patch which projected onto the pixel. This is done after all patches in the environment have been considered. This means that, in the example, if no patch lies behind patches p_1 and p_2, the remaining quarter of the delta form factor will be assigned to p_2.

Errors

The proposed method can be compared to the original hemicube method in the assignment of delta form factors to patches.

For a hemicube pixel covered completely by the closest patch overlapping it, both methods make no errors. Additionally, our normalization scheme ensures that there is no error for a pixel covered by two patches. This is desirable since most hemicube pixels are expected to be covered by a small number of patches.

```
for each patch j from front to back with respect to i (use the BSP tree) {
   project patch j onto hemicube around patch i and
   perform area scan conversion - store result in A
   for each hemicube pixel (f,a,b) {
   (* f is used for the different faces of the hemicube *)
     A_E = A(f,a,b) (1 - e_f(f,a,b))
     F(i,j) = F(i,j) + ΔF(f,a,b) (A_E/A_p)
     e_f(f,a,b) = e_f(f,a,b) + A_E/A_p
     if A_E > 0 then N(f,a,b) = j
     (* N holds the last patch which overlapped each hemicube pixel *)
   }
}
(* Normalization:  *)
for each hemicube pixel (f,a,b) {
   F(i,N(f,a,b)) = F(i,N(f,a,b)) + (1 - e_f(f,a,b)) ΔF(f,a,b)
}
```

Figure 5.4: Algorithm for determination of form factors from a patch i

Note that pathological cases can be constructed for the original hemicube method. Consider, in case of the original method, the situation when there are very small patches visible from the center of a pixel and large patches behind them covering most of the area of the pixel. In this case, the proposed method assigns the delta form factors correctly, whereas the original method would be completely incorrect, i.e., all the delta form factors are assigned to the wrong patches. The new method gives a completely incorrect result only when there are infinitely many infinitesimally small patches.

5.2.4 The Radiosity Solution

Our form factor algorithm places no restriction on the form of the radiosity solution. The solution of the radiosity matrix can be carried out at the same time as form factor determination as in progressive refinement radiosity [Cohen et al., 1988] and substructuring as discussed in [Cohen et al., 1986] can also be performed to take care of the other problems with the hemicube method.

The result can now be rendered and displayed; the BSP tree generated as the preprocess can be used to speed up the rendering process.

5.3 Implementation and Results

A fairly general implementation of this algorithm has been done. The input is given as a list of *convex* polygonal faces; any concave faces will have to be partitioned into convex pieces before being input. The input faces are divided into patches in a simple manner using a threshold size. More intelligent patch generators can be used to replace this part of the implementation. A BSP tree of the patches is then generated (the order of generation of patches and BSP trees has been reversed in this implementation). The form factor calculation can be done using both the original hemicube and the new algorithm. This is followed by a solution of the radiosity matrix and rendering of the scene from a given viewpoint and direction.

As a test case, we consider the situation in Figure 5.5. The only reflecting surface is the floor and there is only one light source (which is non-reflecting). Therefore, any undersampling of the patches of the floor will show up directly as a characteristic aliasing pattern. The patch radiosities on the floor as obtained by the old and the proposed methods (with the same hemicube resolution) are shown in Figures 5.6 and 5.7. Figures 5.8 and 5.9 show the floor as it would be rendered after the radiosity solution. The image obtained from the original hemicube method (Figure 5.8 clearly shows dark bands due to aliasing; these are absent in Figure 5.9.

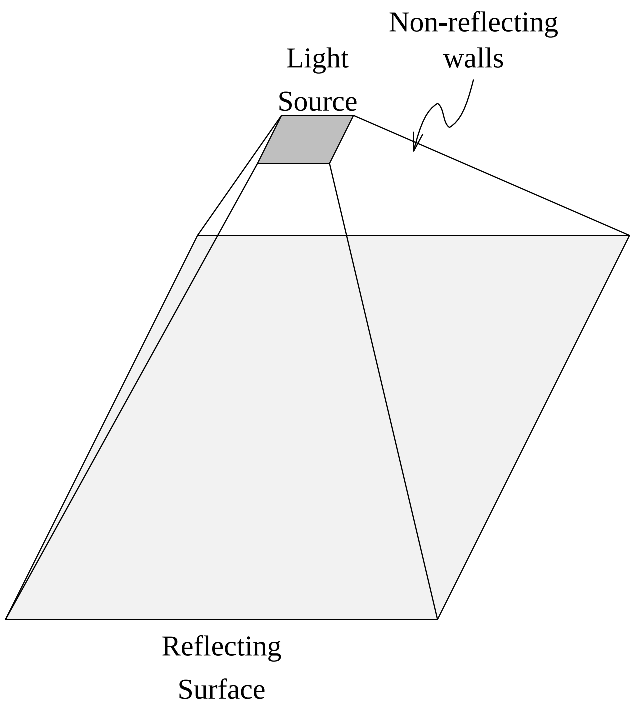


Figure 5.5: A simplified test scene

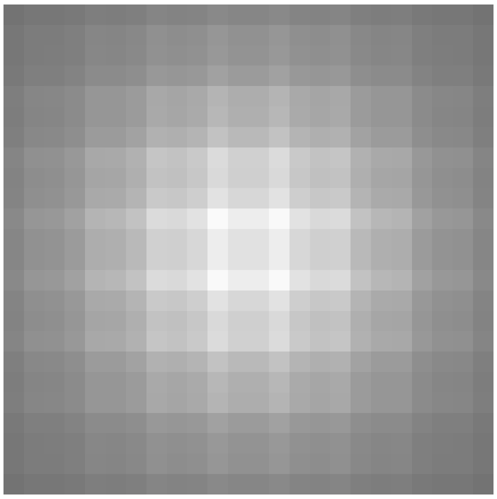


Figure 5.6: Patch radiosities obtained by the hemicube method

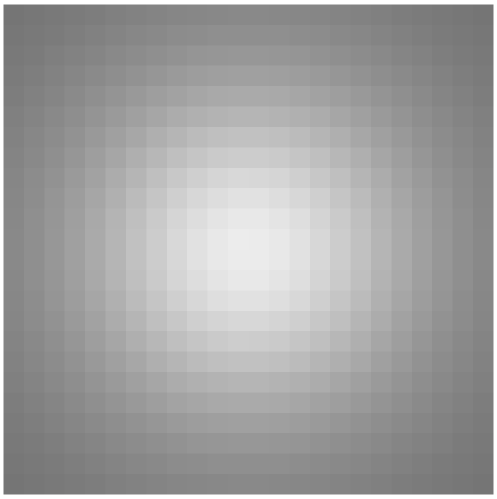


Figure 5.7: Patch radiosities obtained by the proposed method

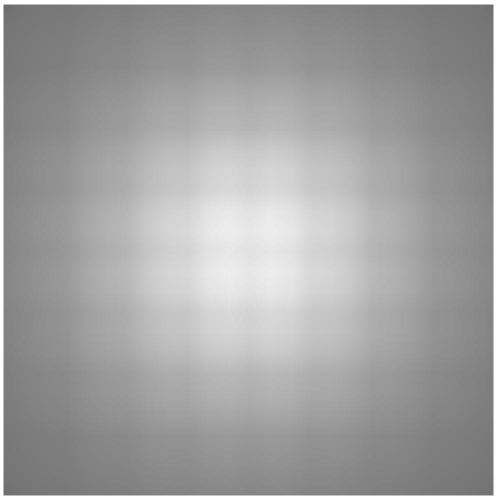


Figure 5.8: Intensity distribution on the floor after bilinear interpolation (original hemicube method)

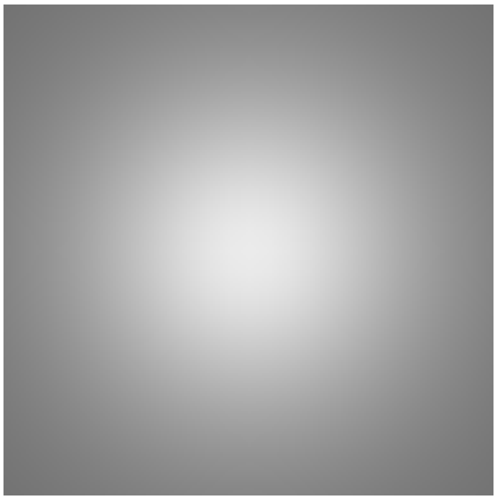


Figure 5.9: Intensity distribution on the floor after bilinear interpolation (proposed method)

Chapter 6

Conclusion

We have seen that the radiosity method is based on the principles of radiative transfer unlike most other illumination models. Of the models currently in existence, it provides the greatest realism, especially in case of diffuse environments. Though it is very expensive, various enhancements have been made to the basic algorithm to make it more efficient and more general by extending it to non-diffuse environments. Form factor determination is the most expensive part of the radiosity algorithm. The hemicube method is one of the more efficient means of calculating form factors.

The new radiosity algorithm we have proposed and implemented substantially reduces the aliasing inherent in the hemicube method. The extra space cost for the algorithm is minimal. The item buffer for storing patch numbers (at every hemicube pixel) in the original hemicube method is replaced here by a "fraction buffer" and the only extra space is required for normalization; we need to store the last patch which projected onto each pixel. The extra computational cost incurred is mainly due to the fact that we need to calculate areas covered during scan conversion of polygons and accounting for occlusion takes more time than a simple comparison in case of the z-buffer in the original hemicube method. The time taken for the preprocess (BSP tree construction) is negligible compared to the time taken by the rest of the radiosity algorithm.

This research was primarily motivated by the fact that one of the major reasons for radiosity not being used extensively today is its cost compared to other methods especially when highly visible aliasing artifacts are completely undesirable. The method discussed in this paper reduces this defect substantially without incurring much of the cost that would be associated with increasing resolution. Further, this extension of area anti-aliasing can be used in situations other than form factor determination. An earlier algorithm which used area anti-aliasing in the image rendering context was proposed in [Catmull, 1978]. This algorithm had very severe costs because it required the determination of actual areas of pixels covered by objects in the image. We get around this problem by dealing with expectation values. In doing so, we have traded error in intensity values for a reduced algorithmic cost.

References

[Baum et al., 1989] Baum, D. R., Rushmeier, H. E., and Winget, J. M. (1989). Improving radiosity solutions through the use of analytically determined form factors. *Computer Graphics*, 23(3):325–334.

[Campbell and Fusell, 1990] Campbell, A. T. and Fusell, D. S. (1990). Adaptive mesh generation for global diffuse illumination. *Computer Graphics*, 24(4):155–164.

[Catmull, 1978] Catmull, E. (1978). A hidden surface algorithm with anti-aliasing. *Computer Graphics*, 12(3).

[Cohen et al., 1988] Cohen, M. F., Chen, S. E., Wallace, J. R., and Greenberg, D. P. (1988). A progressive refinement approach to fast radiosity image generation. *Computer Graphics*, 22(4):75–84.

[Cohen and Greenberg, 1985] Cohen, M. F. and Greenberg, D. P. (1985). The hemicube: A radiosity solution for complex environments. *Computer Graphics*, 19(3):31–40.

[Cohen et al., 1986] Cohen, M. F., Greenberg, D. P., Immel, D. S., and Brock, P. J. (1986). An efficient radiosity approach for realistic image synthesis. *IEEE Computer Graphics and Applications*, 6(2):26–35.

[Fuchs et al., 1980] Fuchs, H., Zvi, M. K., and Naylor, B. (1980). On visible surface generation by a priori tree structures. *Computer Graphics*, 14(3):124–133.

[Goral et al., 1984] Goral, C. M., Torrance, K. E., Greenberg, D. P., and Battaile, B. (1984). Modeling the interaction of light between diffuse surfaces. *Computer Graphics*, 18(3):213–222.

[Hanrahan et al., 1991] Hanrahan, P., Salzman, D., and Aupperle, L. (1991). A rapid hierarchical radiosity algorithm. *Computer Graphics*, 25(4):197–206.

[Immel et al., 1986] Immel, D. S., Cohen, M. F., and Greenberg, D. P. (1986). A radiosity method for non-diffuse environments. *Computer Graphics*, 20(4):133–142.

[Kay and Kajiya, 1986] Kay, T. L. and Kajiya, J. T. (1986). Ray tracing complex scenes. *Computer Graphics*, 20(4):269–278.

[Paterson and Yao, 1989] Paterson, M. S. and Yao, F. F. (1989). Binary partitions with applications to hidden-surface removal and solid modeling. *Proceedings of the Fifth Annual Symposium on Computational Geometry*, pages 23–32.

[Rushmeier and Torrance, 1990] Rushmeier, H. E. and Torrance, K. E. (1990). Extending the radiosity method to include specularly reflecting and translucent materials. *ACM Transactions on Graphics*, 9(1):1–27.

[Sillion et al., 1991] Sillion, F. X., Arvo, J. R., Westin, S. H., and Greenberg, D. P. (1991). A global illumination solution for general reflectance distributions. *Computer Graphics*, 25(4):187–196.

[Wallace et al., 1987] Wallace, J. R., Cohen, M. F., and Greenberg, D. P. (1987). A two pass solution to the rendering equation: A synthesis of ray tracing and radiosity methods. *Computer Graphics*, 21(4):311–320.

[Wallace et al., 1989] Wallace, J. R., Elmquist, K. A., and Haines, E. A. (1989). A ray tracing algorithm for progressive radiosity. *Computer Graphics*, 23(3):315–324.

[Watt and Watt, 1994] Watt, A. and Watt, M. (1994). *Advanced Animation and Rendering Techniques: Theory and Practice*. Addison-Wesley Publishing Company.