Development of a Software Package for Radiosity Computation

by

Wong Kam Wah

A thesis
presented to The University of Hong Kong
in fulfillment of the
thesis requirement for the degree of
Master of Philosophy
in
Computer Science
March 1996

Development of a Software Package for Radiosity Computation

by

Wong Kam Wah

A thesis
presented to The University of Hong Kong
in fulfillment of the
thesis requirement for the degree of
Master of Philosophy
in
Computer Science
March 1996

Abstract of thesis entitled

Development of a Software Package for Radiosity Computation

Submitted by

Wong Kam Wah

for the degree of Master of Philosophy

at The University of Hong Kong

in March 1996

The radiosity method is a newly developed technique for rendering pictures in computers. It uses a large number of small patches to represent the surfaces of a scene and accurately models the interactions of light between the patches according to the laws in physics. The method requires a lot of computations but generates very realistic images of in-door scene with soft lighting. The original version of the radiosity method proposed in 1984 was very slow and required tremendous amount of storage. Since then, many radiosity algorithms which run faster, require less storage, or produce more accurate images have been published. During the same period, the graphics machines become more powerful and cheaper. The objective of this project is to develop a software package on contemporary graphics machines which provides user friendly graphical interface for scene description, automatically divides large surfaces into small patches, and performs radiosity computation.

We investigated in detail some of the relevant algorithms published, used color-coded pictures to visualize the errors induced by those algorithms and identified their sources. By merging the advantages from different algorithms, we have developed an improved radiosity algorithm which concerned mainly on the efficiency and accuracy. To achieve high speed, this algorithm makes use of sophisticated speed-up methods, such as modified BSP tree traversal algorithms and fast ray tracing visibility test. Together with fully utilizing the graphics hardware, such as the depth-buffer and Gouraud interpolation, this algorithm runs four times faster than the other methods. We have also developed another algorithms which is tailored for adaptive environment subdivision. This algorithm improves the quality of picture generated by automatically subdividing surfaces along the shadow boundaries in the environment. Lastly, a user friendly graphical interface is implemented, which makes this package a complete rendering software using the radiosity method.

TABLE OF CONTENTS

CHAPTER 1

Introduction

Nowadays most computer graphics hardware supports illumination calculations. The intensities are usually computed based on simple lighting models, including ambient light, diffuse reflection and specular reflection. However, most of these calculations have limitations. Firstly, they only provide point light sources and parallel light sources. Light sources with area are not supported. Secondly, there is an underlying assumption that every light source is visible to all surfaces in the environment. Thus, no shadow is generated. Thirdly, the interactions of light between diffuse surfaces are barely considered using a global ambient term.

In order to generate more realistic images, the physical behavior of visible light must be modeled in a more detailed way. Goral et al. [15] and Nishita and Nakamae [20] proposed a rendering method called *radiosity*. The original idea was inspired by the heat exchange models in thermal engineering. In an in-door scene consisting of diffuse surfaces, the interactions of light between surfaces are modeled using a set of linear equations. This method accurately takes into account of the interactions of light between diffuse surfaces and generates very realistic images for in-door scenes with area light sources.

The goal of this project is to develop a software package which computes illumination using the radiosity method. A key issue is to find an efficient and accurate algorithm for radiosity computation. Firstly, some of the relevant algorithms published were investigated in detail. By merging ideas from different algorithms, we developed an improved radiosity algorithm which makes use of the nowadays graphics hardware and sophisticated speed-up methods. As a result, this algorithm runs four times faster than the other methods. We have also developed another algorithm which performs accurate surface subdivision automatically. This algorithm gives better picture quality with accurate shadow boundaries, at the same time does not require the users to concern about the low-level implementation details. Lastly, the ideas were implemented into a user friendly software package which runs on high performance computer graphics platforms.

In Chapter 2, a review on some of the radiosity algorithms published in the literature is given. Some of the important concepts in the radiosity method, such as the significance of radiosity and form-factor, are explained in detail.

The quality of the image generated using the radiosity method depends on the accuracy of form-factors calculated. The violations of the underlying assumptions in various methods induce errors in the form-factors computed. In order to visualize the magnitude and orientation of these errors and identify their sources, three of the most widely used methods on computing the form-factors have been studied: the hemicube method [8], Wallace's ray tracing method [30], and Baum's analytical form-factor formula [3]. The errors induced by their methods were displayed using color-coded images. These pictures gave us strong clues to the major causes of the errors. The visualization process and the results are described in detail in Chapter 3.

In Chapter 4, we describe the improved radiosity algorithm in detail. This algorithm is based on Cohen's progressive refinement framework [10] described in Chapter 2. By combining the advantages of the hemi-cube method, Wallace's ray tracing method and Baum's analytical form-factor formula, the algorithm runs four times faster than their methods without sacrificing the quality of the pictures produced.

We have also developed a new version of the radiosity method which produces accurate radiosity solutions by subdividing surfaces in the environment automatically. It satisfies those users who would like to pay a higher cost (in terms of computational time and memory used) in order to obtain better picture quality. The pictures produced will have accurate shadow boundaries, at the same time the users do not need to concern about the low-level implementation details. This algorithm is given in detail in Chapter 5.

The last phase of this project was to make the radiosity rendering method available to end users. Four modules, namely input, tuning, radiosity engine and output, have been implemented for enhancing the applicability of the software package. The input module can import files in one of the common scene description formats which are carefully chosen. The tuning module provides graphical interface for the users to specify the parameters needed for the radiosity calculations. The radiosity engine performs the computations. The output module conforms the results of the radiosity calculations according to Open Inventor [32] format so that the results can be used for various purposes, such as producing pictures, animations, or for interactive architectural walk-through. The system design issues of the entire package are described in Chapter 6. In Chapter 7, a brief summary of this project is given.

CHAPTER 2

A Review of the Radiosity Method

2.1. Basic Concepts

Radiosity is defined as the amount of energy leaving a surface per unit time per unit area. It is the sum of the rate at which the surface emits energy and the rate it reflects the energy received from the environment. Assume that each patch in the environment is an ideal diffuse surface. For surface i, its radiosity can be expressed as in [15]:

$$B_i = E_i + \rho_i \sum_j B_j F_{j\text{-}i} \frac{A_j}{A_i}$$

where

B_i = radiosity of patch i (W/m^2)
E_i = emittance of patch i (W/m^2)
A_i = area of patch i (m^2)
ρ_i = reflectivity of patch i
$F_{j\text{-}i}$ = the fraction of energy leaving patch j that arrives at patch i

Using the relation that $F_{j\text{-}i}A_j = F_{i\text{-}j}A_i$, the interaction of light among all the n patches in the environment can be expressed as a set of linear equations:

$$\begin{bmatrix} 1-\rho_1 F_{1\text{-}1} & -\rho_1 F_{1\text{-}2} & \cdots & -\rho_1 F_{1\text{-}n} \\ -\rho_2 F_{2\text{-}1} & 1-\rho_2 F_{2\text{-}2} & \cdots & -\rho_2 F_{2\text{-}n} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & & \cdot \\ -\rho_n F_{n\text{-}1} & -\rho_n F_{n\text{-}2} & \cdots & 1-\rho_n F_{n\text{-}n} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \cdot \\ \cdot \\ \cdot \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \cdot \\ \cdot \\ \cdot \\ E_n \end{bmatrix}$$

An approximation of the solution of the equations can be obtained using Gauss-Seidel iteration.

The most critical step in the radiosity method is the computation of *form-factors*. The form-factor from small surface (called *patch*) i to patch j, denoted as $F_{i\text{-}j}$, is defined as the fraction of energy leaving patch i that arrives at patch j. The illumination of a receiving patch due to a source patch depends on the radiosity of the source and the form-factor from the source to the patch. Based on the assumption that all patches are ideal diffuse (Lambertian) surfaces, the form-factor between two differential areas dA_i and dA_j (Figure 2.1) is found to be:

$$F_{dA_i\text{-}dA_j} = \delta_{ij} \frac{\cos\theta_i \cos\theta_j dA_j}{\pi r_{ij}^2} \tag{1}$$

where

δ_{ij} = 1 if dA_j is visible to dA_i, 0 otherwise.

The derivation of Equation 1 can be found in [15].

The form-factor between two finite areas A_i and A_j can then be obtained by applying double integration over $F_{dA_i\text{-}dA_j}$:

$$F_{Ai\text{-}Aj} = F_{i\text{-}j} = \frac{1}{A_i} \int_{A_i} \int_{A_j} \delta_{ij} \frac{\cos\theta_i \cos\theta_j dA_j dA_i}{\pi r_{ij}^2}$$

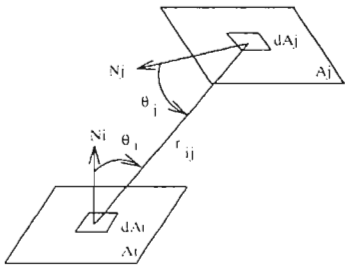


Figure 2.1. Geometry for From-Factor Derivation

It is noticed that the form-factor formula involves a double integral and thus is difficult to compute in a straightforward way. Moreover, the computation of form-factors involves visibility determination. As a result, it is the most time consuming step in the radiosity method. In addition, a matrix of form-factors must be computed and stored even though most of them have little effect on the quality of the final image. Many methods have been suggested for estimating form-factors efficiently. These methods will be described in the next subsection.

Apart from the estimation of form-factors, many works have been done on improving the efficiency of the radiosity method. Cohen et al. [10] have developed a progressive refinement algorithm which avoids the calculation of the entire form-factor matrix. The algorithm proceeds in iterations. In each round, a bright patch whose light energy has not yet been taken into account is chosen to *shoot* its light out. The form-factors from this patch to all the others are calculated and the intensities of the patches are updated. Instead of computing and storing n^2 form-factors, this method only requires to keep track of n form-factors in each iteration. Furthermore, it converges very fast to the accurate solution, and the scene being processed can be displayed before an accurate solution is found.

2.2. Algorithms for Form-Factor Computation

Many methods have been suggested for estimating form-factors efficiently. The hemi-cube method proposed by Cohen et al. [8] is one of the most widely used methods. The key assumption of the method is that two patches are far away from each other, so all the energy emitted from the source can be thought as emitting from the center of the source. The form-factor between the two patches can then be approximated by the form-factor between a differential area at the center of the source and the destination, which reduces the double integrals in the form-factor formula into a single integral. A numerical method is suggested for the integration in the formula. Moreover, the visibility between the source and a patch is determined by the visibility between the source center and the patch, The depth-buffer technique for hidden surface removal can be used in this simplified case.

When two patches are close to each other, the errors in the form-factors computed using the hemi-cube method can be very large. Baum et al. [3] has proposed to use an analytical form-factor formula for the evaluation of form-factors between differential area dA_j and source patch i. This formula assumes that there is no blocking between the source and the differential area. Otherwise, the source patch has to be repeatedly subdivided until all the subsources are either completely visible or invisible from dA_j. The form-factor $F_{dA_j\text{-}A_i}$ is then equal to the sum of the form-factors from dA_j to all visible subsources.

Wallace et al. [30] has suggested a different method for computing form-factors. Light emitted or reflected by a source patch is assumed to be emitting from several circular disks which are evenly distributed on the patch. The visibility of a differential area and a disk is determined by casting a ray from the center of the disk to the area. The form-factor of the receiving patch is computed by considering all the light received

from the visible disks. Instead of computing the radiosities at the centers of patches and then obtaining the vertex intensities using interpolation [8], the radiosities at the vertices of patches are computed directly in this method. The intensities at the other positions of a patch are obtained later by making use of the interpolating facility provided by the graphics hardware. However, ray tracing is a slow process. Moreover, there is an aliasing problem appeared in Wallace's algorithm due to the circular disks sampling method used.

2.3. Algorithms for Environment Subdivision

In order to have accurate shadows in the output, the straightforward radiosity method requires proper subdivisions of the polygons according to the shadow boundaries. Such requirement is usual a big burden on a user as a lot of effort and good knowledge of the implementation details of the radiosity method are needed. Usually, the polygons in the scene are over subdivided, which leads to inefficiency in later computation. It is highly desirable that the subdivisions of surfaces for obtaining accurate shadows are determined automatically. Cohen et al. have proposed an adaptively subdividing scheme [9] which computes the gradient of radiosity over every patches. A patch is recursively subdivided until its radiosity gradient is less than a threshold value. The method improves the quality of a picture without increasing the size of the form-factor matrix.

Campbell et al. [5] used a set of point light sources to approximate an area light source. This method works well for small area light sources. For larger sources, the number of point light sources needed is high, and the shadow boundaries created are too fine.

Lischinski et al. [17] have proposed a discontinuity meshing method for radiosity algorithm. The method classifies the shadow boundaries in an environment into three types: (i) sharp shadow boundaries induced by edges or vertices of occluders lying on the receiving surface; (ii) edge-vertex (EV) events, where the soft shadow boundaries are caused by edges or vertices of occluders that do not touch the receiver but intervene between it and the light source; (iii) edge-edge-edge (EEE) events, where shadow boundaries are caused by the edges of more than two occluders. To locate the shadow boundaries caused by the EV events, they stored the environment in a BSP tree [13] and the tree is used to obtain a front-to-back ordering of surfaces with respect to each sources' vertices and edges. Surfaces are then clipped by the shadow volumes formed by the occluders between itself and the light sources. The EEE events are too complicated to be found out by their method. After all shadow boundaries are found on a surface, the surface is triangulated and the radiosity of receiving vertices are updated.

Chin [7] has developed a similar algorithms to locate the soft shadow boundaries. The environment is also stored in a BSP tree. For each light source being shot, a front-to-back ordering of the receiving surfaces is obtained. The shadow volumes cast by the first surface in the ordering are created. These shadow volumes are represented by two BSP trees: the penumbra BSP tree represents the penumbra area of the shadows, and the umbra BSP tree represents the umbra area of the shadows. The second surface in the ordering is clipped by these two shadow volumes, and the two BSP trees are then enlarged by merging the second surface's shadow volumes. The third surface in the ordering is processed and so on. This method is fast, and the shadow boundaries can be determined accurately. However, it suffers from a major problem: in order to have a correct front-to-back ordering, if a light source intersects with any surface's extending plane, the light source is needed to be divided. In a complex environment, this will cause

too many subdivisions of light sources. As a result, the clipping of receiving surfaces are far too fine.

Many researchers have studied how to illuminate a scene with both diffuse and specular reflection [16,24,25,27,31]. These works make the radiosity method become an important technique in rendering. However, the enhancement of the radiosity method with specular refection is not the focus of this project.

CHAPTER 3

Visualization of Errors on Form-Factor Computation

The quality of the image generated using the radiosity method depends on the accuracy of form-factors calculated. The violations of the underlying assumptions in various methods induce errors in the form-factors computed. Our objective was to visualize the magnitude and orientation of these errors and identify their sources. In addition, guidelines for avoiding large errors in applying these methods are proposed.

We have studied three of the most widely used methods for computing form-factors: the hemicube method, Wallace's ray tracing method, and Baum's analytical form-factor formula. Attention was paid to the normalized distance between the source and the destination, the aspect ratio of the source patch, and the aliasing effects.

3.1. Methods for Form-Factor Computation

The hemi-cube method proposed by Cohen et al. [8] is one of the most widely used methods. The key assumption of the method is that two patches are far away from each

other, so all the energy emitted from the source can be thought as emitting from the center of the source. The form-factor between two patches can then be approximated by the form-factor between a differential area at the center of the source and the destination. Moreover, the visibility between the source and a patch is determined by the visibility between the source center and the patch. Based on this assumption, the form-factor between two patches is:

$$F_{A_i-A_j} \approx F_{dA_i-A_j} = \int_{A_j} \delta_{ij} \frac{\cos\theta_i \cos\theta_j dA_j}{\pi r_{ij}^2} \tag{2}$$

A numerical method is suggested for the evaluation of the integration in Equation 2. It is observed that when two patches are projected onto a surrounding surface of a source, if their projections occupy the same area and location, the form-factors between the source and the patches will be equal [8]. Therefore, patches are projected onto a hemi-cube placed above the center of the source. The cube is discretized into small elements called pixels. The form factor between the source center and a pixel, denoted dF, is called *delta form-factors*. The delta form-factors are pre-computed. The form-factor from source patch i to destination patch j is then equal to the sum of dF associated with the pixels covered by the projected image of patch j. Visibility of patches are determined using the depth-buffer technique for hidden surface removal. As projections and depth-buffer operations are supported by most graphics hardware, the hemi-cube method runs efficiently in graphics machines. This approach really makes the radiosity method practical in complex environment.

A disadvantage of the hemi-cube method is that the basic assumption, i.e., the source and destination is far away from each other, does not always hold. When the source and

destination are too close, the form-factor computed is incorrect (Figure 3.1 and Figure 3.2). The normalized distance in Figure 3.2 is the distance between the source and the destination, divided by the area of the source. A more detailed discussion on the accuracy of the hemi-cube method can be found in [3].

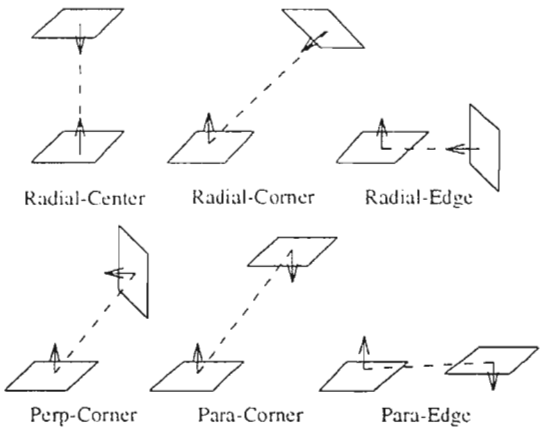


Figure 3.1. Orientations between the source and receiving patch.

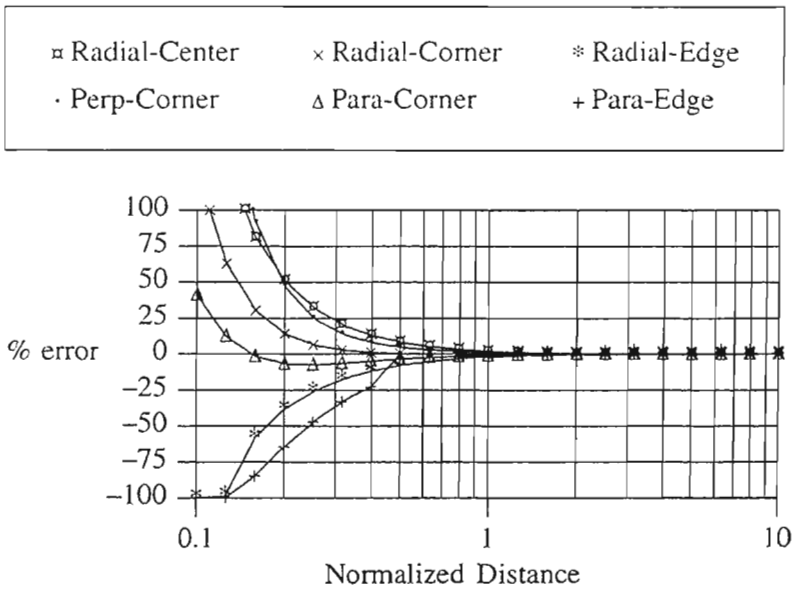


Figure 3.2. Relative errors in the form-factors computed using Equation 2 (both the source and the receiving patch are of aspect ratio 1:1).

The resolution of the hemi-cube also affects the accuracy of the form-factors calculated. When the resolution is low, artifact caused by the aliasing effect is found in the image generated. Figure 3.3 shows the relative errors of form-factors due to the aliasing effect. The source has the same aspect ratio and size as the one in Figure 3.1, and the area of the destination is one-fourth of the area of the source. The only difference is that Figure 3.2 is obtained by using Equation 2 directly, whereas Figure 3.3 is obtained by placing a hemi-cube over the source. The resolution of the top plane of the hemi-cube is 500×500. Note that when the destination patch is far away from the source, its projection may only partially overlap with one or few pixels. Thus, the relative errors caused by low resolution are indeed magnified when the patches are far from each other. Fortunately, when a patch is far away from a source, its form-factor is small and a large relative error will not seriously affect the quality of the image.

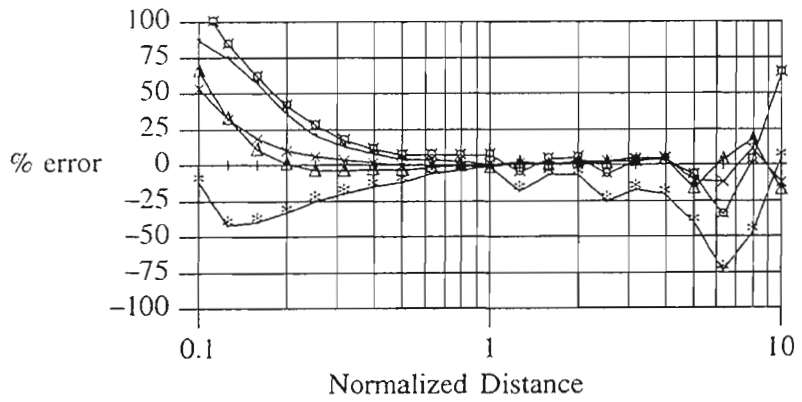


Figure 3.3. Relative errors in the form-factor values computed by the hemi-cube method.

When two patches are close to each other, the errors in the form-factors computed using the hemi-cube method can be very large. Baum et al. [3] has proposed to use an analytical method for the evaluation of delta form-factors when two patches are closer

than a threshold value. The formula of form-factor between differential area dA_j and patch i as shown in Figure 3.4 is:

$$F_{dA_j\text{-}A_i} = \frac{1}{2\pi} \sum_{g \in G_i} N_j \cdot \Gamma_g \tag{3}$$

where

G_i is the set of edges in patch i;
N_j is the normal of differential surface dA_j;
Γ_g is a vector with magnitude equal to the angle γ (in radians) and direction equal to the cross product of the vectors R_g and R_{g+1} shown in Figure 3.4.

The derivation of Equation 3 can be found in [26].

This formula assumes that there is no blocking between the source and the differential area. Otherwise, the source patch is subdivided until all the subpatches are either completely visible or invisible from dA_j. The form-factor $F_{dA_j\text{-}A_i}$ is then equal to the sum of the form-factors from dA_j to all visible subpatches.

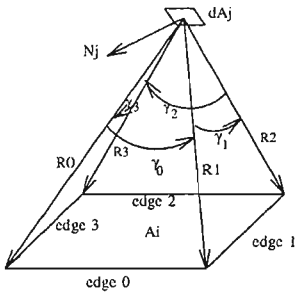


Figure 3.4. Geometry for evaluating analytical form-factor

Wallace et al. [30] has suggested a different method for computing form-factors. Light emitted or reflected by a source patch is assumed to be emitting from several circular disks which are evenly distributed on the patch. The visibility of a differential

area and a disk is determined by casting a ray from the center of the disk to the area. The form-factor of the receiving patch is computed by considering all the light received from the visible disks.

The form-factor between circular disk k and differential area dA_j (Figure 3.5a) can be approximated as:

$$F_{A_k\text{-}dA_j} \simeq \frac{dA_j \cos\theta_j \cos\theta_k}{\pi r^2 + A_k} \tag{4}$$

The form-factor between source patch i and vertex j (Figure 3.5b) is:

$$F_{A_i\text{-}dA_j} = dA_j * \frac{1}{N} \sum_k \delta_k \frac{\cos\theta_{jk} \cos\theta_{ik}}{\pi r_k^2 + \frac{A_i}{N}} \tag{5}$$

where

N = number of sample points used on the source patch
δ_k = 1 if the ray can reach sample point k;
0 otherwise

Instead of computing the radiosities at the centers of patches and then obtaining the vertex intensities using interpolation [8], the radiosities at the vertices of patches are computed directly in this method. The intensities at the other positions of a patch are obtained later by making use of the interpolating facility provided by the graphics hardware.

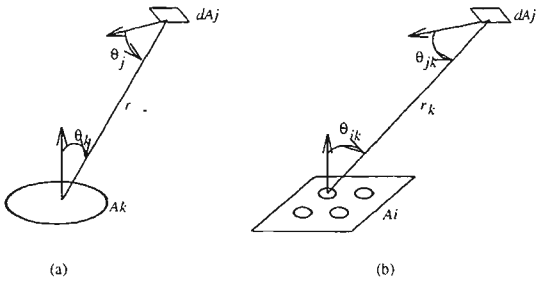


Figure 3.5. a) Geometry between an arbitrarily oriented disk and a differential area.
b) Geometry between a source patch (approximated by multiple disks) and a differential area.

When a receiving vertex is far away from a source, the errors induced by Equation 4 and 5 are negligible. However, when a receiving vertex is close to the surface of a source patch, this method induced relatively large errors. The errors are caused by two levels of approximation. The first level is the approximate formula from a circular disk to a vertex (Equation 4). Figure 3.6 shows the errors of the form-factors obtained using Equation 4.

Distance from source to receiving area (r)	1R	5R	10R
Average relative errors of Wallace's formula (Equation 4)	55.38%	5.55%	1.44%
Max. relative errors	99.97%	11.38%	2.96%

Figure 3.6. Errors induced by Wallace's formula over the range $0 \leq \theta_k \leq 89$, $0 \leq \theta_j \leq 45$. R is the radius of the source. (θ_k, θ_j are defined in Figure 3.5.)

We have derived a new approximate formula for estimating the form-factor from a disk to a differential area:

$$F_{k\text{-}j} \approx \frac{dA_j \cos\theta_j \cos\theta_k}{\pi r^2 + A_k - 2\pi r R \sin\theta_k} \times \frac{\sqrt{r^2 + R^2 - 2rR\sin\theta_k}}{\sqrt{r^2 + R^2 + 2rR\sin\theta_k}} \tag{6}$$

A detailed derivation can be found in the appendix. The computation of form-factors using this formula runs as fast as Wallace's one, but more accurate. The errors of this formula is shown in Figure 3.7.

Distance from source to receiving area (r)	1R	5R	10R
Average relative errors of Equation 6	32.71%	1.13%	0.28%
Max. relative errors	98.25%	4.0%	1.0%

Figure 3.7. Error analysis of the new approximate form-factor formula.

The second level of approximation occurs when Wallace's algorithm treats a polygonal subsource as a circular disk. The shape of the source patch is not equal to the union of the circular disks. Some regions of the source are not covered and some regions are covered more than once. As a result, aliasing effect is induced. Wallace et al. also have mentioned this kind of errors in their paper [30]. They suggested to subdivide the source adaptively for each receiving vertex such that more sample disks are placed near the vertex if it is too close from the source.

3.2. Errors in Environment Without Blocking

In this section, errors induced by the three methods for form-factor computation are studied. For comparison purpose, it is assumed that there is a light-emitting patch i in the environment (Figure 3.8). The various methods are applied to calculate the energy received by a very small patch dA_j in the vicinity of patch i. The area of patch i is one unit and it emits one unit of energy per unit time (i.e., $B_i A_i = 1$). The area of patch dA_j

is 0.0001 unit and it always points at the center of patch i. The formulas of the energy received by patch dA_j using the three different methods are given below. Note that in this particular case, the form factor is equal to the energy received and the analytical method gives the exact value.

(1) The Hemi-Cube method (HC)

$$B_i A_i F_{i\text{-}dA_j}$$

$$= \frac{\cos\theta_c \cos\theta_j dA_j}{\pi r_{cj}^2} \qquad \textit{by Eq. 2 and Eq. 1}$$

$$= \frac{\cos\theta_c dA_j}{\pi r_{cj}^2}$$

where c is the center of patch i, and r_{cj} is the distance between c and j.

(2) The Analytical formula (ANA)

$$B_i A_i F_{i\text{-}dA_j}$$

$$= F_{i\text{-}dA_j}$$

$$= F_{dA_j\text{-}i} * \frac{dA_j}{A_i}$$

where $F_{dA_j\text{-}i}$ is given by Equation 3.

(3) The Ray Tracing method (RT)

$$B_i A_i F_{i\text{-}dA_j}$$

$$= dA_j * \frac{1}{N} \sum_{k=1}^{N} \frac{\cos\theta_{ik} \cos\theta_{jk}}{\pi r_{jk}^2 + \frac{A_i}{N}} \qquad \textit{by Eq. 5}$$

where N is the number of sample points. The distributions of 3 or 9 sample points on source patch of various aspect ratio are shown in Figure 3.9.

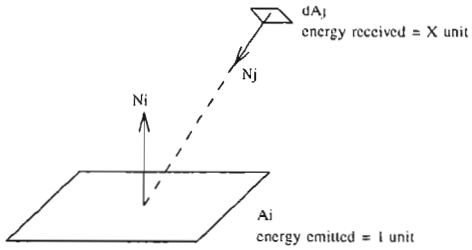


Figure 3.8. Energy received at a point over a light source.

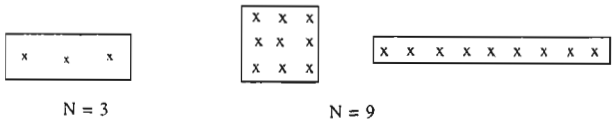


Figure 3.9. Distribution of sample points on source patches.

To visualize the errors, the "isoenergy" surface around the source patch is displayed. In Figure 3.10, the locations where patch dA_j receives 0.8E-5 unit of energy is drawn. The aspect ratio of the source patch is 1:1. The left one is the correct answer which is obtained using the ANA method. The right one is obtained using the HC method. As the size and shape of the two surfaces are similar, the errors induced by the HC method are negligible in this case.

The displays in Figure 3.11 are obtained in the same way except that the energy received is raised to 3.0E-5 unit. The isoenergy surfaces obtained are closer to the source patch. Note that the HC method overestimates the energy received right above the center of the source patch, and underestimates the energy received around the edges of the source patch. This is due to the assumption of the HC method that energy is emitted from the center of the source patch. In Figure 3.12, the isoenergy surfaces of the same

energy level as Figure 3.10 are displayed for a source patch of aspect ratio 1:3. As the HC method does not take into account of the shape of the source patch, there is no surprise that the errors increase when the aspect ratio of the source patch becomes larger.

Figure 3.10. Isoenergy surface where 0.8E-5 unit of energy received.

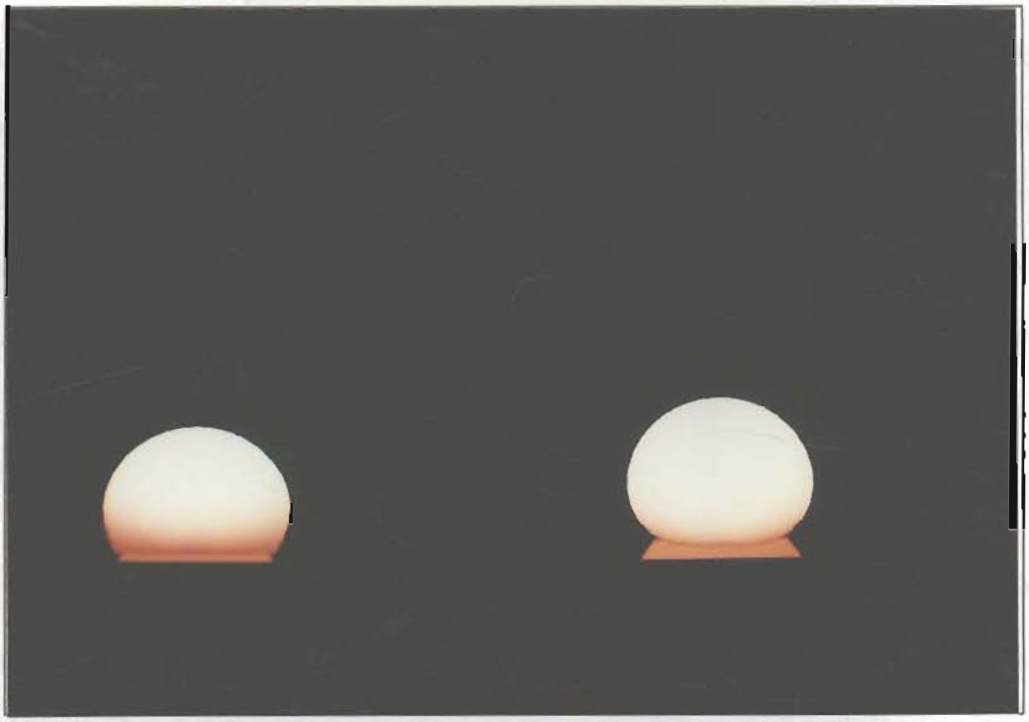


Figure 3.11. Isoenergy surface where 3.0E-5 unit of energy received.

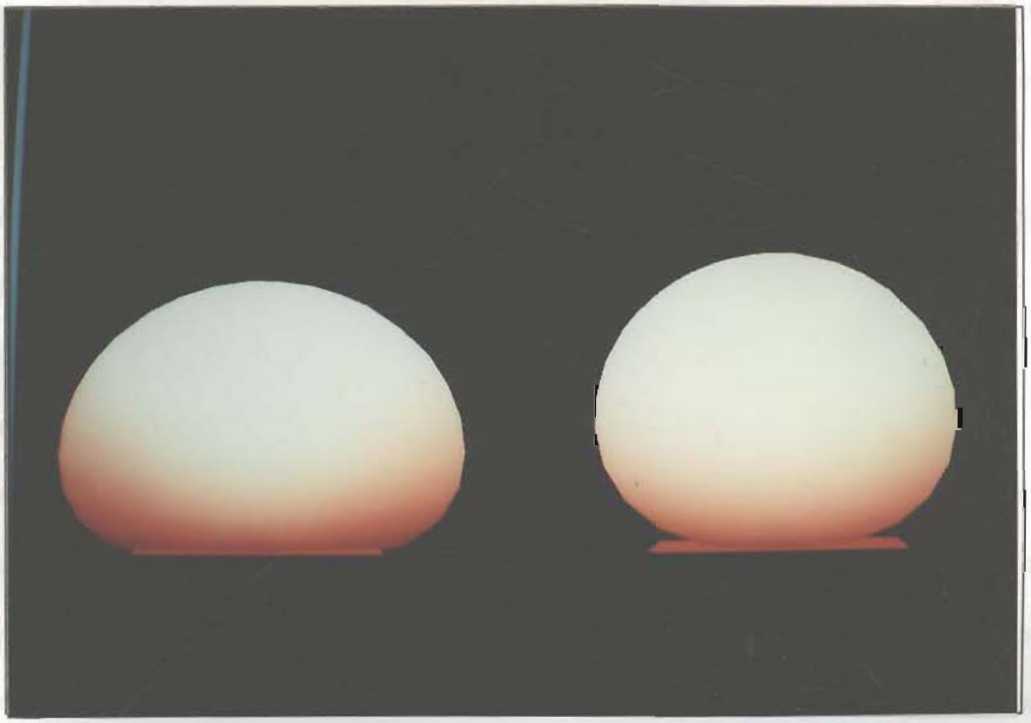


Figure 3.12. Isoenergy surface where 0.8E-5 unit of energy received, with source patch aspect ratio 1:3.

In order to study how the errors induced by the HC method are affected by the distance, the energy received by patch dA_j at different locations of a cross section of space above the source patch are computed. The magnitude of the energy is rendered using a color coding. White color is used to represent the highest energy received, and blue color to represent the lowest. The cross section above a source patch of aspect ratio 1:3, together with the color scale ruler, is shown in Figure 3.13. Again, the one on the left is obtained using the ANA method and the one on the right is obtained using the HC method. It is clear that the errors diminish when the distance becomes larger.

The cross sections are computed again using the RT method. The result is shown in Figure 3.14, with a source patch of aspect ratio 1:3. The left one is obtained using the ANA method and the right one is obtained using the RT method with three sample points. The right one is quite similar to the correct one on the left hand side. However, aliasing is observed at the positions very close to the surface of the source patch. In Figure 3.15, the cross sections for a source patch of aspect ratio 1:9 is drawn. The top one is obtained using the ANA method; the pictures in the bottom row are obtained using the RT method with three and nine sample points, respectively. The result shown in the bottom left picture is not acceptable as too few sample points are used. Also, the aliasing effect in these drawings is significant. Further study shows that the distribution of the sample points plays an important role in the magnitude of the aliasing effect. As a general guideline, the sample points should be evenly distributed and each represents the radiosity of a squarely region of the source patch. Moreover, it is found that the RT formula (Equation 4) may underestimate the amount of light by as much as 15% at some positions close to the source patch [33]. This explains why the bottom displays are not as bright as the upper one, especially near the surface of the source.

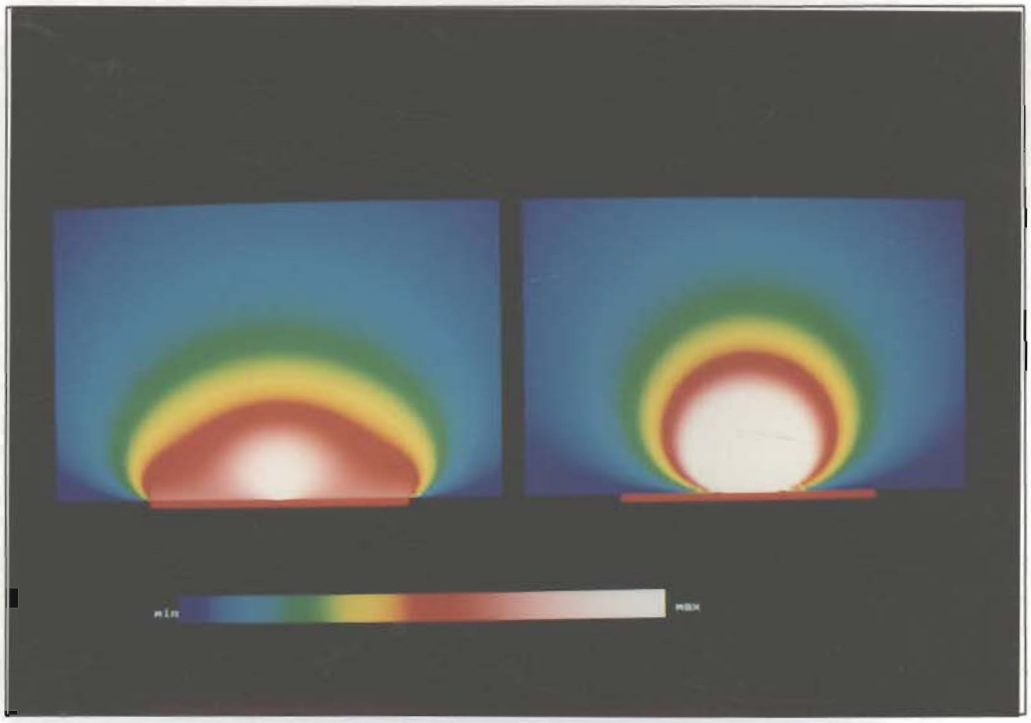


Figure 3.13. Cross section energy diagram obtained by the HC method.

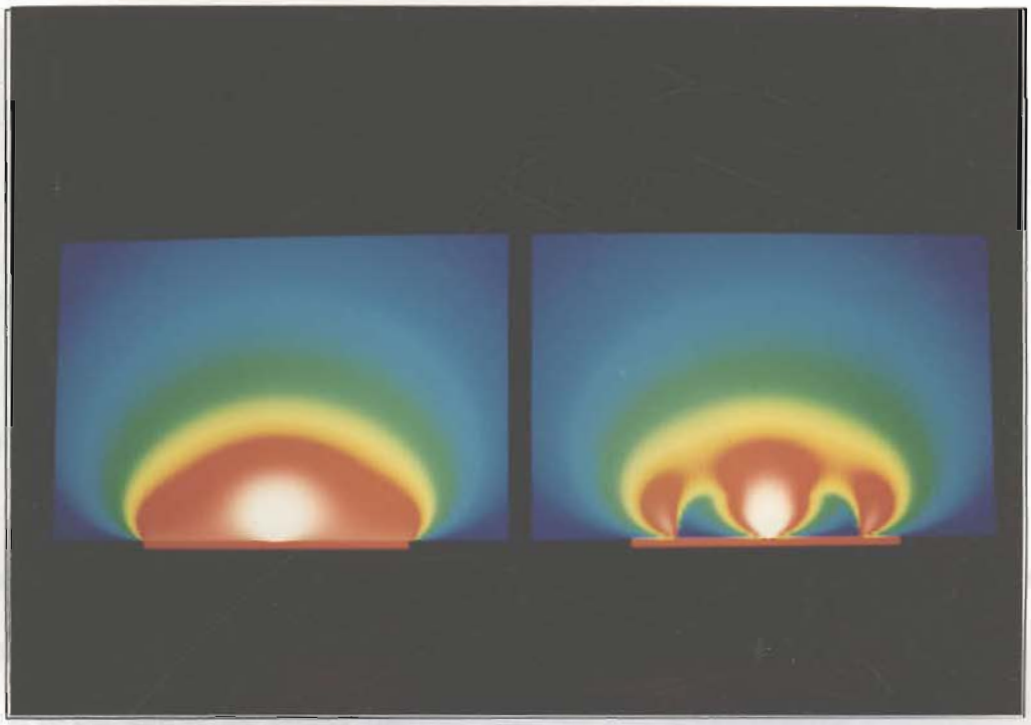


Figure 3.14. Cross section energy diagram obtained by the RT method.

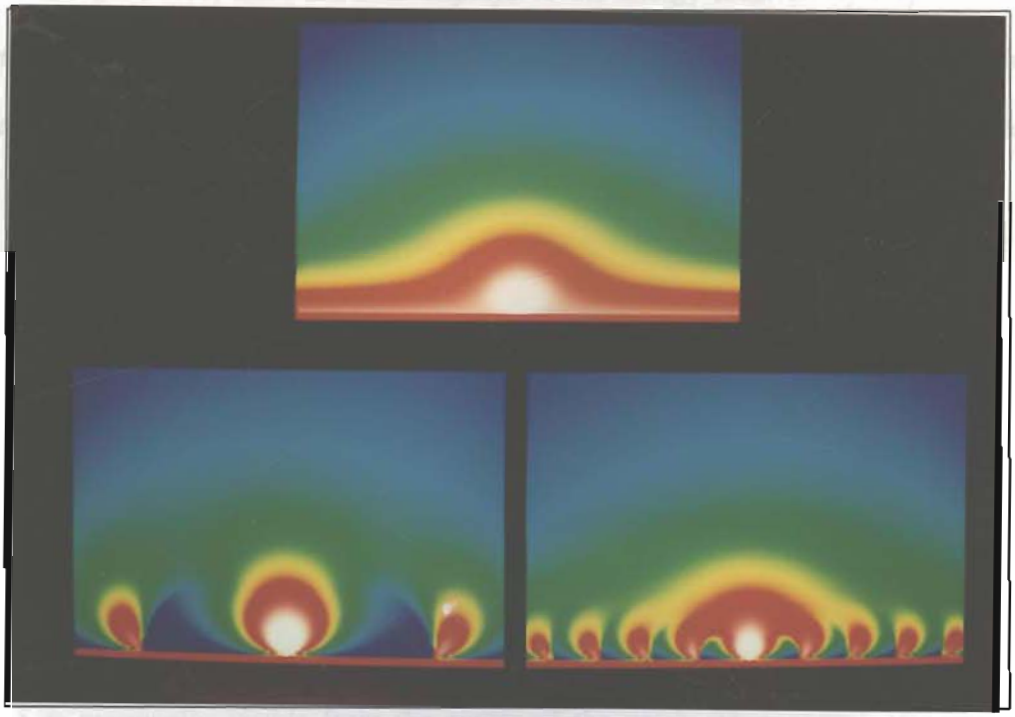


Figure 3.15. Cross section energy diagram obtained by the RT method, with source patch aspect ratio 1:9.

3.3. Errors in Environment With Blocking

In an environment with blocking, it is necessary to perform visibility test before application of the form-factor formulas described in previous section. In the HC method, the entire patch i is assumed visible to dA_j if the center of dA_j can been seen from the center of A_i. In the RT method, a light-emitting disk is assumed entirely visible to dA_j if the center of dA_j can been seen from the center of the disk. In the ANA method, if patch i is not entirely visible from the center of dA_j, it is subdivided recursively until each component is either completely visible or invisible. The total energy received is then equal to the sum of the energy received from the visible components.

In Figure 3.16, there are four displays of the energy received by dA_j at locations of a cross section of space above patch i. The size of patch i is 1×1 unit area. An obstacle patch of size 0.5×0.5 unit area is placed at 0.25 unit length above the center of patch i.

The upper left display is obtained using the ANA method which gives the correct amount of energy received by dA_j. Note that only a small area right behind the obstacle is completely dark and the amount of energy changes continuously. The upper right is obtained using the HC method. As light is assumed to be emitting from the center, a very large shadow area behind the obstacle is completely dark. Comparing with the upper left, it is obvious that the HC method underestimates the amount of energy in the shadow area. On the other hand, the method overestimates the value in the vicinity of the shadow which is marginally visible from the center of the patch. These errors are caused by the fact that the method fails to take into account of partial blocking in the formulation.

The lower left display in Figure 3.16 is obtained using the RT method with nine sample disks. Note that each disk induces sharp shadow as in the case of the HC method because the entire disk is either *seen* or *unseen* from a location. Discrete change of intensity is observed at the intersections of the shadow boundaries. Nonetheless, this display is closer to the correct one than the display obtained using the HC method.

It is interesting to find out why the RT method is more accurate than the HC method. The study here indicates that it is not due to the use of ray tracing technique for visibility test, nor the use of circular disks in radiosity calculation. The critical factor is that the RT method assumes that light is emitted from **nine** disks on the source rather than from the center of the source as assumed in the HC method. The lower right display of Figure 3.16 is obtained using the HC method after cutting patch i into nine pieces. This display is as good as the one obtained using the RT method.

The displays shown in Figure 3.17 are obtained in the same way as those in Figure 3.12 except that the obstacle is shifted to the right 0.25 unit length. These displays confirm our findings in the above discussions.

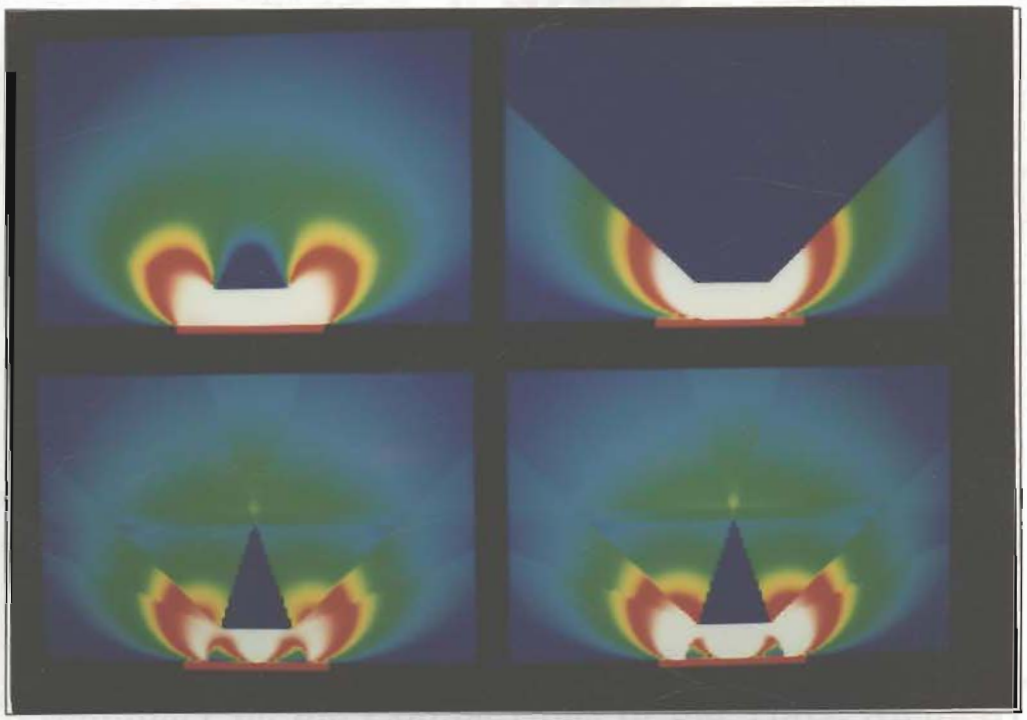


Figure 3.16. Cross section energy diagrams with obstacle.

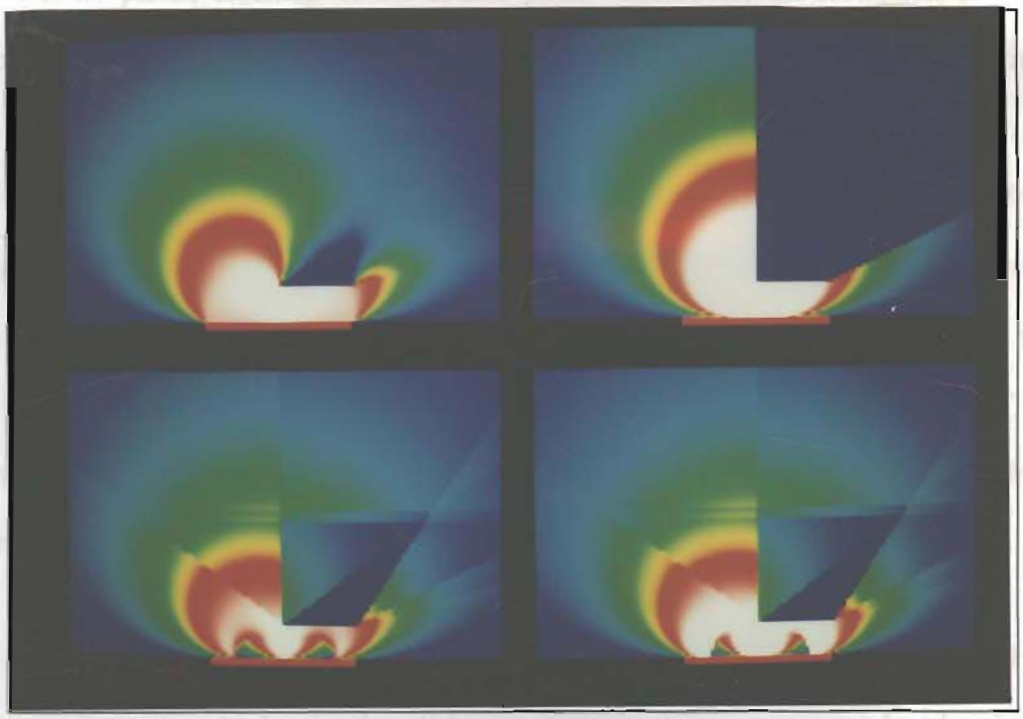


Figure 3.17. Cross section energy diagrams with shifted obstacle.

3.4. Summary on the Comparisons of the Three Methods

The ANA method is used to obtain the exact form-factors in the comparison. The computation time is acceptable if there is no blocking between the source and destination. However, in an environment with blocking, the process of finding the visible portion of the source patches is time consuming. In a complex environment, this process is extremely slow even when sophisticated data structures, such as binary-space-partition trees, are used. Thus, the ANA method is generally not practical.

Due to the assumption that light emits from the center of a source patch, the HC method gives accurate results only if the aspect ratio of the source patch is close to one, the normalized distance between the source patch and the receiving patch is large, and

there is no partial blocking of the source patch. Moreover, the resolution of the hemi-cube has to be high enough. From our experience, the method gives acceptable results if the following conditions are satisfied: (i) the aspect ratio is between 1/2 and 2; (ii) the normalized distance is greater than 0.3; (iii) the amount of light energy emitted from any single subsource is less than 5% of the total light energy emitted in the environment; (iv) the resolution of the hemi-cube surfaces is at least 500×500. A source patch shall be divided if any of the first three conditions is violated. A major advantage of the HC method is that it is by far faster than the other two methods as it can be implemented using the facility supported by graphics hardware.

The RT method generates more satisfactory results than the HC method mainly because it uses multiple circular disks to approximate a source patch. Another major improvement is that the method computes the vertex radiosity instead of the patch radiosity. Such modification enables direct application of the intensity interpolation facility provided by graphics hardware. As ray tracing is a time consuming process, this method in general runs slower than the HC method. Moreover, aliasing is found in locations very close to a source patch. To obtain good results in applying this method, some guidelines similar to those given for the HC method should be followed. For example, each disk is made associated with a region similar to a square; each disk emits no more than 5% of total light energy; the normalized distance between a disk and a vertex should be greater than 0.3.

CHAPTER 4

An Efficient Radiosity Algorithm

Having studied the most popular algorithms for computing form-factors, we come up with the following conclusions: (i) the hemicube algorithm is efficient and simple. However, it suffers from two major limitations. Firstly, aliasing appears in the projected images due to the discrete nature of the hemicube pixels. As a result, the form-factors computed are inaccurate. Secondly, Gouraud shading is commonly used to render the images in most graphics workstations. This shading algorithm makes use of the vertex intensities of the polygon being rendered. However, the hemicube algorithm computes only the patch radiosities. The vertex intensities are obtained by interpolating the patch radiosities. This interpolation may induce another level of errors; (ii) Wallace's ray tracing algorithm overcomes hemicube's problem. However, ray tracing is a slow process. Moreover, there is an aliasing problem with Wallace's algorithm due to the circular disks sampling method in their algorithm.

Base on the progressive refinement [10] framework, we tried to combine the hemicube algorithm and Wallace's ray tracing algorithm. By adding several modifications, a complete radiosity algorithm is proposed which runs much faster than either of the above methods alone without sacrificing the quality of the pictures produced.

Firstly, we observed that ray tracing is not necessary for most vertices. A modified hemicube method can be used to perform a visibility pre-test before the rays are cast. As a result, the number of rays traced is reduced substantially. Secondly, when ray tracing is performed, a binary space partitioning (BSP) tree [13] is used to sort the patches in a front-to-back order according to the orientation of the ray being traced. This order helps us reduce the number of object-ray intersection calculations. Thirdly, there is an aliasing problem with Wallace's algorithm due to the sampling circular disk method used. This problem is solved by replacing the Wallace's form-factor formula with Baum's analytical formula. The resulting algorithm is described in pseudo code below:

```
Input : a scene represented by convex polygons
Output : radiosities of the polygons' vertices

Radiosity ()
{
        pre-processing, such as building the BSP tree;

        for each source of large radiosity do {

                sub-divide the source into several subsources;

                for each subsource do {

                        project all patches onto the hemicube over the subsource
                        and determine the vertices' visibility;

                        for each vertex with ambiguous visibility {

                                perform ray tracing visibility test, using the BSP tree;

                        }
                        for each vertex visible from the subsource {

                                use Baum's analytical formula to compute the form-factor
                                between this vertex and the subsource;

                                update the vertex's radiosity;

                        }
                }
        }
}
```

4.1. Hemicube Visibility Test

In Wallace's algorithm, a light source is divided into several (circular) subsources. The visibility between a subsource and a vertex is determined by ray tracing. Rays are cast from the center of a subsource to all vertices. Although this method determine accurately the visibility between the center of a subsource and a vertex, the ray tracing step is a slow process.

We suggest to perform a hemicube visibility pre-test for each subsource. Before rays are cast from a subsource, a hemicube is placed above the subsource and all the other patches are projected onto the hemicube. Each hemicube pixel covered by a patch's projected image stores the *id* of the patch. When two projected images occupy the same pixel, the *id* of the one which is closer to the subsource is stored.

To determine the visibility of a vertex, we locate the hemicube pixel where this vertex is projected onto. The value (the patch *id*) stored in that pixel is checked against the *id* of the patch containing the vertex. If they are equal, the vertex is visible from the subsource. If they are not equal, the eight neighbouring pixels are checked. When none of them has stored the *id* of the patch containing the vertex, the vertex is invisible from the subsource. Otherwise, the visibility of this vertex is determined by ray tracing.

The eight neighbouring pixels have to be checked for avoiding the aliasing errors. If one of the eight neighbouring pixels has stored the *id* of the patch containing the vertex, the vertex is marginally visible or invisible from the subsource. In this case, ray tracing is needed to determine the visibility.

Results of experiments showed that in most cases, the visibility of a vertex can be determined by the hemicube pre-test without ray tracing. In addition, the hemicube can

be implemented by the graphics hardware. We have implemented the algorithm on a Silicon Graphics IRIS Indigo/XS24 workstation. Two simple scenes were tried, and the time used to shoot the light of one subsource is recorded and shown in Figure 4.1. These data confirm that the hemicube pre-test does speed up the visibility determination process. However, the set-up overhead in the software implementation of the pre-test is not justified when the number of vertices is small.

Number of vertices in the scene	796	4311
pure ray tracing visibility test	1.2	21.6
with hardware hemicube pre-test	1.2	6.8
with software hemicube pre-test	5.3	12.0

Figure 4.1. Time (in sec) used to shoot the light of one subsource.

4.2. Ray Tracing Visibility Test Using a BSP Tree

When a vertex's visibility cannot be determined by the hemicube pre-test, ray tracing visibility test is performed. There are many speed-up methods for ray tracing suggested in literature [4,12]. Since in the radiosity algorithm all the objects are represented with polygonal faces, we propose a new speed-up method which uses a BSP tree [13].

A BSP tree is a binary tree which represents a partitioning of space by planes. Each plane is spanned from a polygon in the scene. The root of the tree is a polygon selected from the scene. This polygon's plane divides the space into two half-spaces. Each subtree of the root node represents a half-space. All the polygons lying in front of the root's plane are assigned to the left subtree of the root. Similarly, all the polygons lying behind the root's plane are assigned to the right subtree of the root. A polygon that lies on both sides of the root's plane is split by the plane and each piece is assigned to the

appropriate half-space. Both the left subtree and the right subtree of the root are constructed recursively in the same fashion. Figure 4.2(a) shows a 2D example of a scene and Figure 4.2(b) shows one possible BSP tree of the scene. Given an arbitrary viewpoint, a modified inorder traversal of the BSP tree provides a front-to-back ordering of the polygons with respect to this viewpoint [6]. Figure 4.2(c) shows the front-to-back order of the polygons obtained by an inorder traversal of the BSP tree guided by the view point *S*. Further details of BSP tree can be found in [6,13].

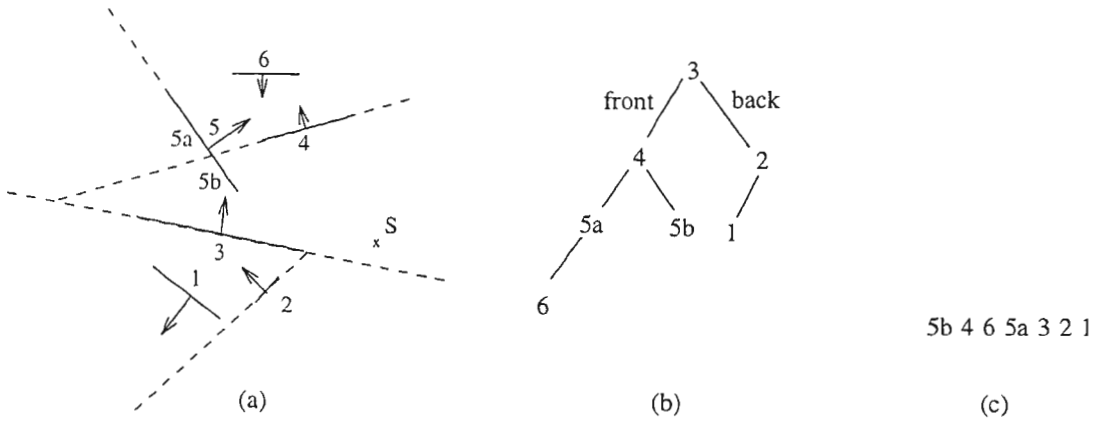


Figure 4.2. (a) A two dimensional scene;
(b) a BSP tree for the scene;
(c) a front-to-back ordering obtained with respect to *S*.

In our algorithm, a BSP tree is built for the input scene at the beginning. In each shooting process, when a ray is cast from a subsource to a vertex, the ray's starting position is used to guide the BSP tree traversal and obtain a front-to-back order of the patches. The object-ray intersection calculations are performed according to the order obtained until the first intersection is found. If the first intersection occurs at the vertex, the vertex is visible from the subsource. Otherwise, it is invisible. Following the order obtained from the BSP tree traversal ensures that the first intersection found is the closest intersection along the ray.

We can use the direction of the ray to modify the BSP tree traversal such that the number of intersection calculations is further reduced. For example, as shown in Figure 4.3(a), a ray starting from point S is being traced. At a certain step the BSP tree traversal will reach the node containing the plane P. Let N_p be the node. Since S lies on the back of the plane P, the ordinary BSP tree traversal will visit the right subtree of N_p first, and then process N_p (i.e. test the patches lying on P for intersection), and finally visit the left subtree of N_p. However, as the ray is going away from P, the receiving vertex will not be on the plane P or in the left subtree of N_p. Moreover, the patches on plane P or in the left subtree of N_p will never block the ray. Therefore, those patches can be ignored. After the right subtree of N_p is visited, the BSP tree traversal can back-track two levels to the parent of N_p. This reduces the number of patches to be tested. Figure 4.3(b) and (c) show the other possible orientation between a plane and a ray, and state the corresponding actions. Empirical results show that this BSP tree speed-up method reduces nearly half of the time needed by the straightforward ray tracing process.

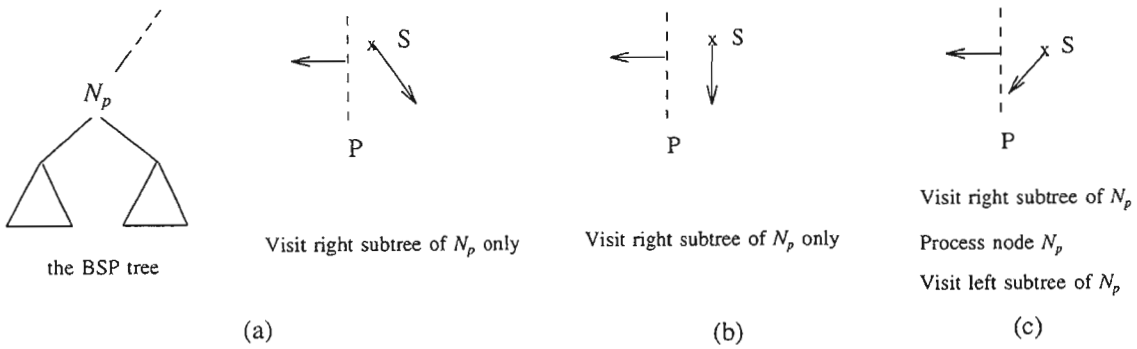


Figure 4.3. Different cases which will be encountered and the corresponding actions during the ray tracing visibility test with a BSP tree.

Note that this BSP tree speed-up method is a special case of the object space subdivision method suggested in the literature [14]. In that method, the object space is equally divided into enough subspaces using the octree until each subspace contains only

a few number of objects. When a ray being traced enters a subspace, only the objects inside the subspace are tested for intersection. If the ray does not hit any objects, the algorithm should find the next subspace into which the ray moves. The process of finding the next subspace is difficult and time consuming. Moreover, using octree to divide the object space usually results in too many subdivisions. In our BSP tree speed-up method, the space subdivision is determined by the geometry of the polygonal faces. Paterson and Yao [23] showed that an $O(n^2)$-sized BSP tree can be built using the randomized method, where n is the number of planar facets.

It should be pointed out that this BSP tree speed-up method is suitable for a scene which contains only polygonal faces. If an object (such as a sphere or a cylinder) is not represented with polygons, the algorithm should use other ray tracing speed-up methods. In a scene which contains only polygonal faces, this BSP tree speed-up method is simple and efficient. Another advantage of this method is that the BSP tree built can be used for other purposes, such as the *adaptive mesh generation* proposed in [5].

4.3. Analytical Form-Factor Formula

For each vertex visible from the subsource being shot, the form-factor between this vertex and the subsource is needed to be computed. In our algorithm, the Baum's analytical formula [3] is used. In this case, the actual shape of the subsource is accurately taken into account

To investigate the execution overhead of using Baum's formula, the form-factors from a square source patch to a differential area in 4140 different orientations and positions are computed. The computation took 0.9 second when Baum's formula was

used. When the source was assumed to be a circular disk and Wallace's formula was used, the computation took 0.1 second. Although the Baum's formula is 9 times slower than Wallace's one, this part of computation is only a small portion in the entire process of the radiosity algorithm.

4.4. Experiments and Results

We have implemented both the original Wallace's algorithm and our improved version. In Figure 4.5, a room is rendered using these two algorithms. It contains 96 patches before subdivision and contains 3642 subpatches after subdivision. Totally 4311 vertices are rendered (because most subpatches share vertices). Pictures on the left column are produced by Wallace's algorithm. Visibility is determined by ray tracing, and the form-factors are calculated by their approximate formula. Pictures on the right column are produced by our algorithm. All the three improvements mentioned in this chapter are implemented. There is no obvious difference between these two set of pictures. Figure 4.4 shows the execution time used for producing these picture. Our algorithm runs about four times faster than the original one.

Number of subsources shot	9	18	33
Time (sec) used by Wallace's algorithm	171.8	302.2	407.8
Time (sec) used by our algorithm	35.4	75.8	107.2

Figure 4.4. cpu times used for producing Figure 4.5.

Figure 4.5. A room after the light of
(a) 1 source (totally 9 subsources) is shot;
(b) 2 sources (totally 18 subsources) are shot;
(c) 5 sources (totally 33 subsources) are shot.

CHAPTER 5

An Adaptive Subdivision Algorithm for Radiosity

5.1. Shadow Generated by Area Light Sources

As mentioned in Chapter 2, in order to have a high quality output, it is highly desirable that the subdivisions of surfaces for obtaining accurate shadows can be performed automatically. Many methods have been published for finding the shadow boundaries in an environment with point light sources [12,6].

In an environment containing area light sources, the determination of shadow boundaries is more difficult. A point in the environment may be either visible to an entire light source, totally invisible from the light source, or visible to a portion of the light source. In the latter case, it is necessary to determine which portion of the area light source is visible from the point. Such computation usually requires a lot of effort.

Many methods for locating the shadow boundaries induced by an area light source make use of the concepts of *penumbra* volume and *umbra* volume. Section 5.2 gives the definitions of these two terms and their significance.

Chin et al. [7] has proposed an algorithm to find the shadow boundaries induced by area light sources. The algorithm makes use of the penumbra and umbra volumes which are represented using BSP trees. The algorithm is described in detail in Section 5.3. A major shortcoming of the algorithm is explained in Section 5.4. A new algorithm which overcomes the shortcoming in Chin's method is described in Section 5.5. This new algorithm runs much faster than the original version. Experimental results on the performance of the improved algorithm are described in Section 5.6.

5.2. Penumbra and Umbra Shadow Volume

Given a point in an environment, it is said to be inside a light source's *umbra* if the light source is completely invisible to it. A point is said to be inside a light source's *penumbra* if the light source is partially visible to it.

For each polygon in the scene, there is a penumbra volume and an umbra volume of this polygon with respect to each area light source. If a point is in the polygon's umbra volume, the point is totally blocked from the light source by this polygon. Similarly, if a point is in the polygon's penumbra volume, the point is partially blocked from the light source by this polygon.

According to [7], given a convex polygon and a convex area light source, if the spanning plane of the polygon does not divide the light source and vice versa, the penumbra and umbra shadow volumes of the polygon with respect to the area light source can be constructed from three kinds of planes. The *scene polygon plane* is the plane of the polygon itself. This plane divides the entire space into two halves: a positive space and a negative space. With the assumption that the area source does not cross a scene

polygon plane, the half space containing the source is defined as the polygon's positive space. The *light-source vertex plane* is defined as the plane spanned by a vertex of the light source and an edge of the polygon. This plane divides the space into two halves and the half space containing the polygon is defined as the light-source vertex plane's negative space. The *light-source edge plane* is defined as the plane spanned by an edge of the light source and a vertex of the polygon. Similarly, the half space containing the polygon is defined as the light-source edge plane's negative space.

The penumbra volume is the intersection of the polygon's negative half-space and the negative half-spaces of certain light-source vertex planes and light-source edge planes. The light-source vertex planes and light-source edge planes are those with the vertices of the light source entirely containing in the plane's positive half-space or on the plane.

The umbra volume, which is entirely contained within the penumbra volume, is the intersection of the polygon's negative half-space with the negative half-spaces of certain light-source vertex planes. The light-source vertex planes are those with the vertices of the light source entirely containing in the plane's negative half-space or on the plane. No light-source edge planes contribute to the umbra volume.

Figure 5.1 gives a two dimensional example showing the penumbra and umbra volumes of a line segment P with respect to a line segment light source S.

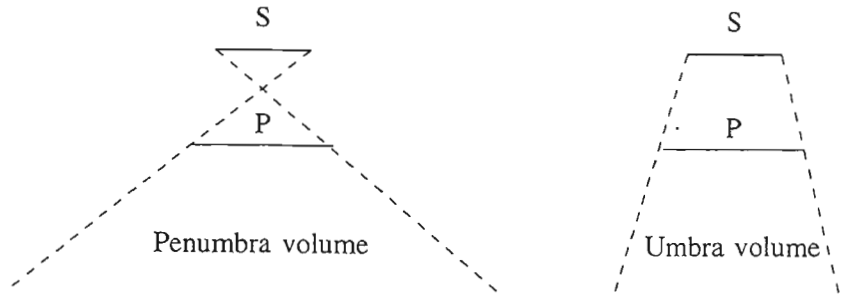


Figure 5.1. The penumbra and umbra volume of line segment P with respect to light source S.

5.3. Finding the Penumbra and Umbra Volume Using BSP Trees

Chin et al. [7] proposed an algorithm to find the shadow volumes induced by area light sources. Their method uses two BSP trees to represent the penumbra and umbra volumes with respect to an area light source. The following gives the pseudo code of a progressive refinement method for radiosity computation which incorporates Chin's Algorithm for guiding the subdivisions.

```
Input : a scene represented by convex polygons
Output : fragmented polygons and radiosities of the polygons' vertices

Radiosity ()
{
        construct a BSP tree containing all polygons in the scene;

        for each light source S do {

                if  S intersects with a scene polygon plane  then
                        divide S by this plane;
        }

        for each resulting light source S,
        in the descending order of light intensity, do {

                obtain a front-to-back ordering of polygons
                with respect to the center of S by using the BSP tree;

                PenumbraTree := UmbraTree := empty;

                for each polygon, in the front-to-back order defined by
                traversing the BSP tree using S, do {

                        divide the polygon according to
                        the PenumbraTree and UmbraTree;

                        update the radiosities of the polygon vertices;

                        Enlarge the PenumbraTree and UmbraTree by taking
                        into account the shadow volumes of the polygon
                        with respect to S;
                }
        }
}
```

At the beginning, a BSP tree is built to store the space partitioning by the planes of all scene polygons. During each iteration in progressive refinement, when a light source is chosen to shoot out its energy, this BSP tree is used to find out a front-to-back ordering of all the polygons with respect to the source's center. For each polygon in the ordering, this method divides it into completely visible, partially visible and completely invisible fragments according to the penumbra and umbra trees built so far. Lastly, the two trees are enlarged by merging themselves with the shadow volumes of the polygon being

processed with respect to the light source being shot.

The penumbra volume and umbra volume are represented with BSP trees in the algorithm. Figure 5.2 is a simple example showing how to represent the shadow volumes with BSP trees. New nodes with a label *out* or *in* are added to an ordinary BSP tree as leaves. An *out* node represents a region which is outside the shadow volume represented by the tree, and an *in* node represents a region inside the shadow volume. To divide a polygon into fragments which are either completely visible, partially visible or completely invisible, the polygon is *filtered* down the tree and clipped by the tree node's planes if it is intersected by the planes. The process is performed recursively until each fragment reaches an *out* node or an *in* node.

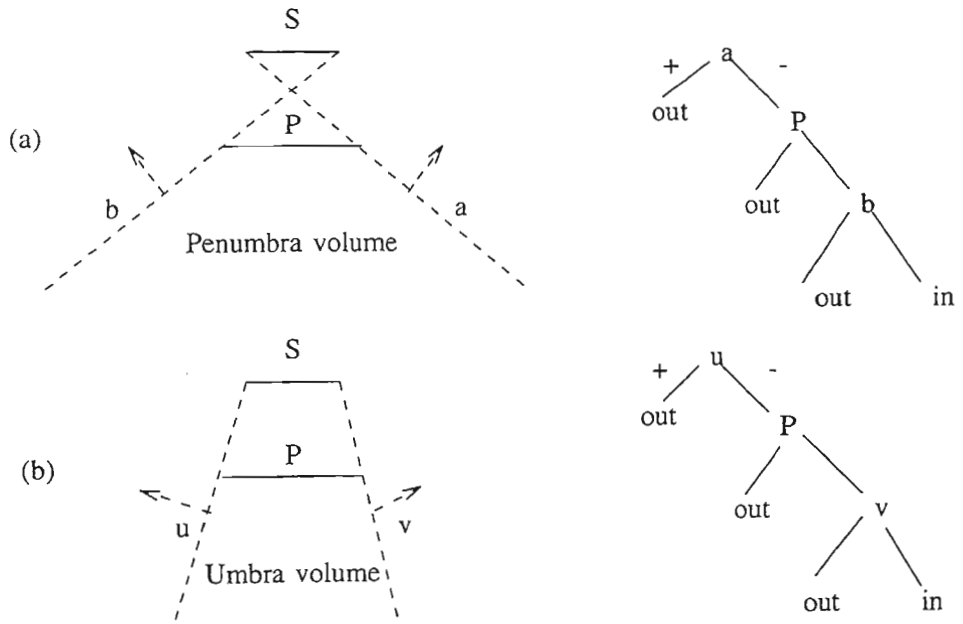


Figure 5.2. (a) A BSP tree representing a penumbra shadow volume; (b) A BSP tree representing an umbra shadow volume.

For the first polygon in the front-to-back ordering, no shadow will be cast on it due to this light source. The algorithm updates the vertices' radiosities of the polygon. After that, the shadow volume cast by this polygon is worked out and is represented by the penumbra BSP tree and the umbra BSP tree. The second polygon in the front-to-back ordering is then divided by these two shadow volume trees for obtaining accurate shadow boundaries. The vertices' radiosities of the polygon are updated accordingly and the two shadow volume trees are enlarged by merging themselves with the shadow volumes cast by the polygon. The third polygon in the front-to-back ordering is then processed and so on.

When a vertex of a patch outside the penumbra volume is completely visible from anywhere on the source, the form-factor between this vertex and the source can be obtained analytically. When a vertex of a patch inside the umbra volume is totally invisible from the source, no energy is received by this vertex from the source. In the remaining case, the source is partially visible from a vertex of a patch inside the penumbra volume but outside the umbra volume. In Chin's algorithm, such a patch is further divided into a grid of small subpatches and the radiosity of each subpatch's vertex is computed. The portion of the area source which is visible to a vertex is determined by traversing the BSP tree of all scene polygons using the vertex as the viewing position. The area light source is filtered down and clipped during the traversal. The light energy received are then computed from the fragments which are visible from the vertex.

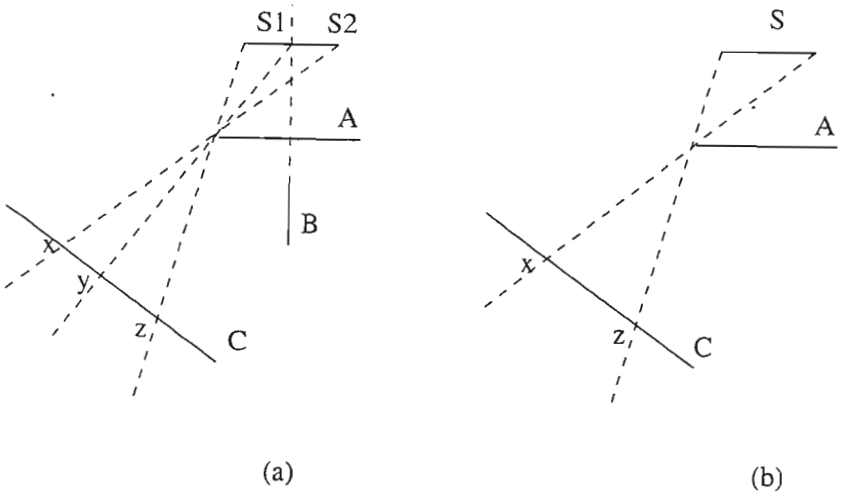
5.4. Major Drawback of Chin's Algorithm

As described in [6], finding shadow volumes using the BSP tree traversal method works well when the environment contains point light sources. When the method is applied to

environment with area light sources, it uses the center of an area light source as the viewpoint to drive the BSP tree traversal. In order to have a correct front-to-back ordering of the polygons with respect to the area light source, a light source which intersects with any scene polygon plane has to be divided into two by the plane and the resulting sources are treated independently. This division step is performed at the beginning of the algorithm.

In a complex environment, an area light source easily intersects with many polygon's planes, and is divided recursively into a very large number of subsources. Consequently, the algorithm runs very slowly.

Figure 5.3 shows a simple example explaining why an environment is over-subdivided by the algorithm. As shown in Figure 5.3(a), since the light source S intersects with the plane spanned by polygon B, it is divided into two subsources, $S1$ and $S2$. The algorithm later divides polygon C at point y, because y is lying on the penumbra shadow boundary formed by $S1$ and surface A. However, as shown in Figure 5.3(b), there is no need to divide C at y because y is not lying on the penumbra shadow boundary formed by S and surface A. The division of S into $S1$ and $S2$ in the preprocessing is actually aimed for the worst situations and such subdivision is not needed in processing most polygons in the scene.



(a) (b)

Figure 5.3. (a) Subdivision of C if S is divided by B; (b) subdivision of C if B does not exist.

5.5. A Fast Area-Driven BSP Tree Traversal

Instead of dividing a light source at the beginning of the algorithm, we proposed to postpone the subdivision until the area light source reaches a BSP tree node associated with a plane intersects with it. In each iteration of the progressive refinement, the area light source chosen is used to guide the BSP tree traversal. The situations are exactly the same as that of a point light source if the area light source does not intersects with the plane associated with the BSP tree node being visited. When the traversal reaches a node *P* whose plane intersects with the light source, the light source is divided into two accordingly. Let the subsource in the positive space of the plane be the positive subsource and the one in the negative space be the negative subsource. The traversal is now forked into two and each guided by one of the subsource. The order of the traversal guided by the positive subsource is: positive subtree of *P*, *P* and then the negative subtree of *P*. The order of the traversal guided by the negative source is just the opposite. The

two traversals are performed independently. The division of the source is no longer needed when the two traversals are complete. The source is treated as one piece again when the traversal backtracks to the parent of P.

As described in the previous sections, two shadow volumes are enlarged during the traversal of the BSP tree. In the modified algorithm, when a light source reaches a tree node whose plane intersects with the source, the two shadow volumes are duplicated and enlarged independently in the two traversals forked. When both traversals complete, the two penumbra volumes are merged into one again, and so is the two umbra volumes. In our implementation, we use the method described in [29] for merging two BSP trees representing volumes.

Figure 5.4 shows a two dimensional scene with an area light source representing by a rectangular bar. When the original algorithm is used, the light source are divided into three. The traversal sequences are shown in Figure 5.4(c). When the new algorithm is used, the light source S is divided only when the traversal reaches node *5a*. The source is considered as a single piece again when the traversal backtrack from node *5a*. The next subdivision occurs when node *2* is traversed. Figure 5.4(d) shows the resulting traversal sequences. Note that the number of nodes traversed and processed is less than that in the original algorithm. Therefore, much work is saved and the new algorithm runs faster than the original one.

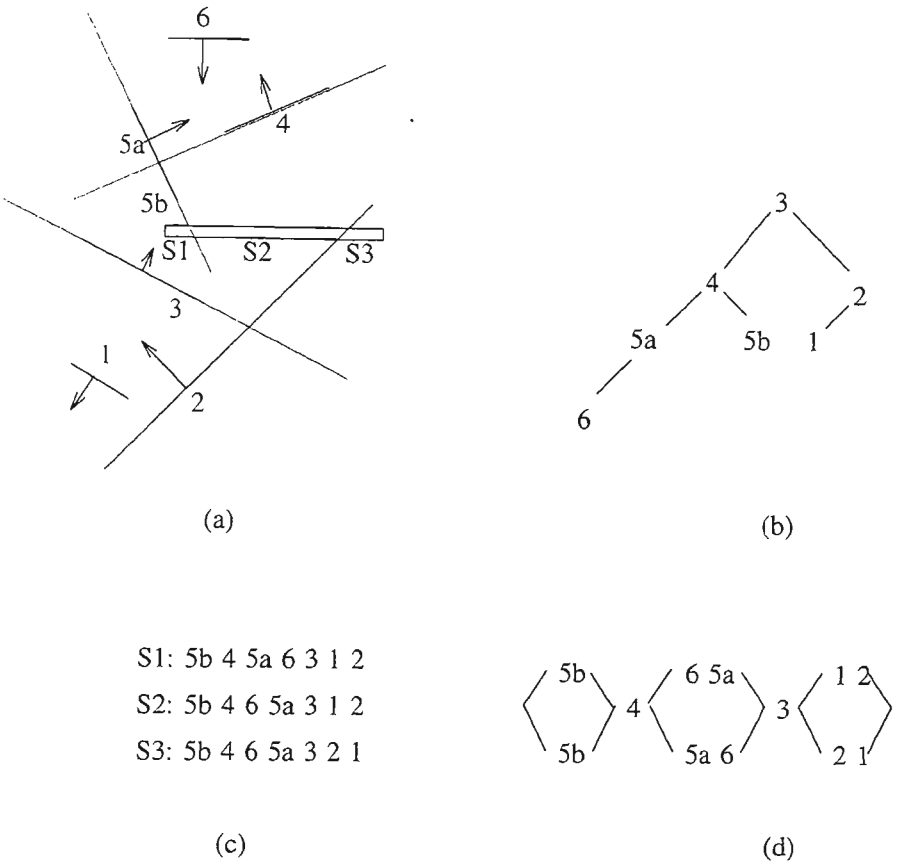


Figure 5.4. (a) A two dimensional scene; (b) A BSP tree for the scene; (c) The traversal ordering in the original algorithm; (d) the traversal ordering in the new algorithm.

To avoid the inefficiency in finding the visible portion of a source from a vertex in the penumbra region, the energy received by the vertex is approximated by sampling. A set of evenly distributed points on the source is first chosen. The visibility of each point from the vertex is determined using the improved visibility test described in previous sections. The ratio of the number of visible points over the total number of points approximates the ratio of the energy received by the vertex over that in the case of no blocking. The latter can be computed using Baum's analytical formula described earlier. Experiments show that this sampling method speeds up the algorithm without noticeable

defects in the resulting images.

A pseudo code description of the modified algorithm is given as follow:

```
Input : a scene represented by convex polygons
Output : fragmented polygons and radiosities of the polygons' vertices

Radiosity ()
{
        Construct a BSP tree containing all polygons in the scene;

        for each light source s, in the order of descending intensity, do {
                PenumbraTree := UmbraTree := empty;
                Shoot( s, PenumbraTree, UmbraTree, Btree->root );
        }
}

Shoot ( surface src, PenumbraBSP ptree, UmbraBSP utree, BSPTreeNode P )
{
        if (P == NIL) return;

        if ( src lies on P->plane's positive side ) {

                Shoot ( src, ptree, utree, P->positiveChild );
                For each polygon F lies on the plane of P {
                        use ptree and utree to find the shadow boundaries on F;
                        update the radiosities received by the vertices' of F
                                and its subpatches;
                        enlarge ptree and utree by merging them with the
                                shadow volumes cast by F;
                }
                Shoot ( src, ptree, utree, P->negativeChild );

        } else if ( src lies on P->plane's negative side ) {

                /* similar to above */

        } else {
                Divide ( src, P->plane, src_pos, src_neg );
                TreeCopy ( ptree, pt1 ); TreeCopy ( utree, ut1 );
                TreeCopy ( ptree, pt2 ); TreeCopy ( utree, ut2 );
                Shoot ( src_pos, pt1, ut1, P );
                Shoot ( src_neg, pt2, ut2, P );
                ptree = mergeTree ( pt1, pt2 );
                utree = mergeTree ( ut1, ut2 );
        }
}
```

5.6. Experiments and Results

To evaluate the improvement of the new algorithm over the original one, two simple scenes as shown in Figure 5.5 are rendered. Figure 5.4 shows the statistics of the experiments.

In the first scene, an area light source is located directly above the box. In the second scene, an area light source is located above the chair. When the original algorithm is used, the source is divided into 6 pieces by the sides of the box in the first scene and is divided into 16 pieces by the sides of the chair's legs in the second scene. In the new algorithm, the light source is divided only when necessary and much replicated works is saved. As a result the algorithm runs much faster than the original one. The amount of time saved is proportional to the total number of subdivisions of light sources in the original algorithm.

Note that the pictures produced by the improved algorithm are indistinguishable from those produced by the original method. Although the new algorithm subdivides less polygons than the original one, there is no deteriorating of the quality of the pictures produced. It is because the speed-up is due to the saving of the redundant subdivisions of polygons in the original algorithm.

box	polygon input	polygon output	sub light sources	time (min)
improved algorithm	90	23599	1	7.5
original method	90	36184	6	16

chair	polygon input	polygon output	sub light sources	time (min)
improved algorithm	43	33924	1	6.5
original method	43	64516	16	116

Figure 5.4. cpu times used for producing Figure 5.5.

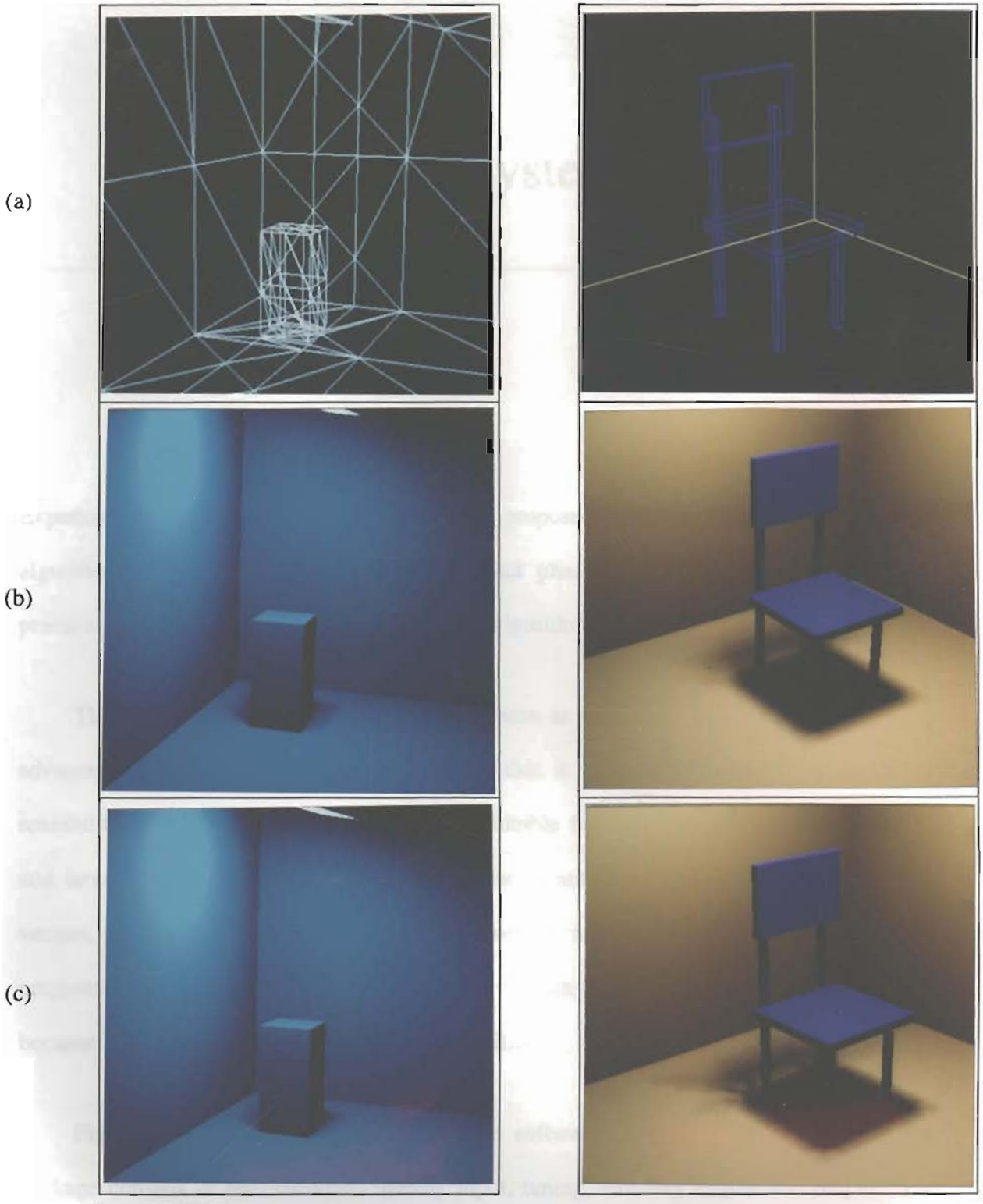


Figure 5.5. Two simple scenes shown by:
(a) wireframe pictures before computation;
(b) results produced by original algorithm;
(c) results produced by our algorithm.

CHAPTER 6

System Design Issues

Experiments and evaluations showed that the proposed algorithms run faster than other algorithms published in the literature. The last phase of the study is to implement a practical software package using the proposed algorithms.

The object-oriented methodology was chosen as the programming framework. The advantage of object-oriented programming is that it enhances the programming code's reusability and extensibility. It is especially suitable for computer graphics programming and large development tasks. Many fundamental entities in computer graphics, such as vectors, planes and polygons, can be implemented as *objects* and be used in various programs. Among many object-oriented programming languages, C++ [28] was chosen because of its efficiency, flexibility, and portability.

Figure 6.1 shows the architecture of the software package. The core part of the package consists of four modules: namely input, tuning, radiosity engine and output. The input module reads in a file which contains information about the surfaces in the environment being processed. The tuning module provides user with controls of the radiosity computations. The radiosity engine performs computations. The output module

saves the results into a file for display. The detailed functionalities of each module are described in the following sections.

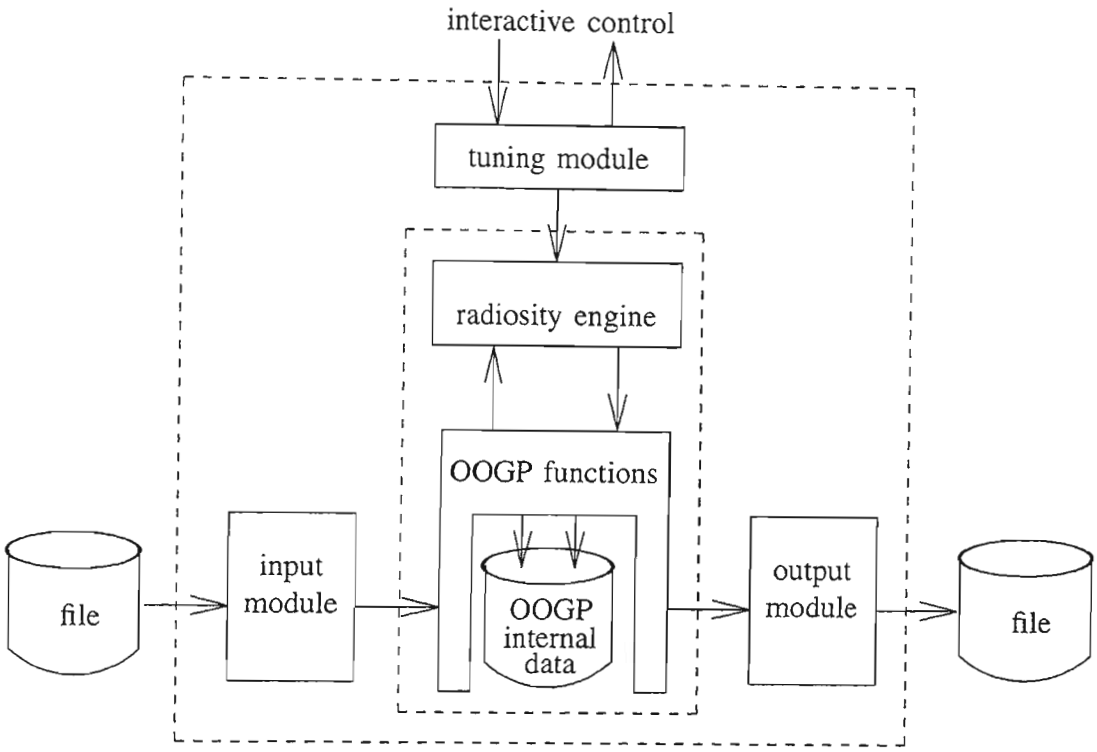


Figure 6.1. Block diagram of the radiosity software.

6.1. The Kernel -- an Object-Oriented Graphics Package

The most fundamental component of this package is a collection of *objects*, grouped as a programming library called the object-oriented graphics package (OOGP). OOGP consists of C++ data structures together with functions for their manipulation. Many commonly used computer graphics entities, such as vectors, lines, planes, vertices and polygons are included. The use of these objects is not limited to radiosity computation, they are also frequently used in many other computer graphics applications. Figure 6.2

shows the class hierarchy of the objects in the OOGP programming library.

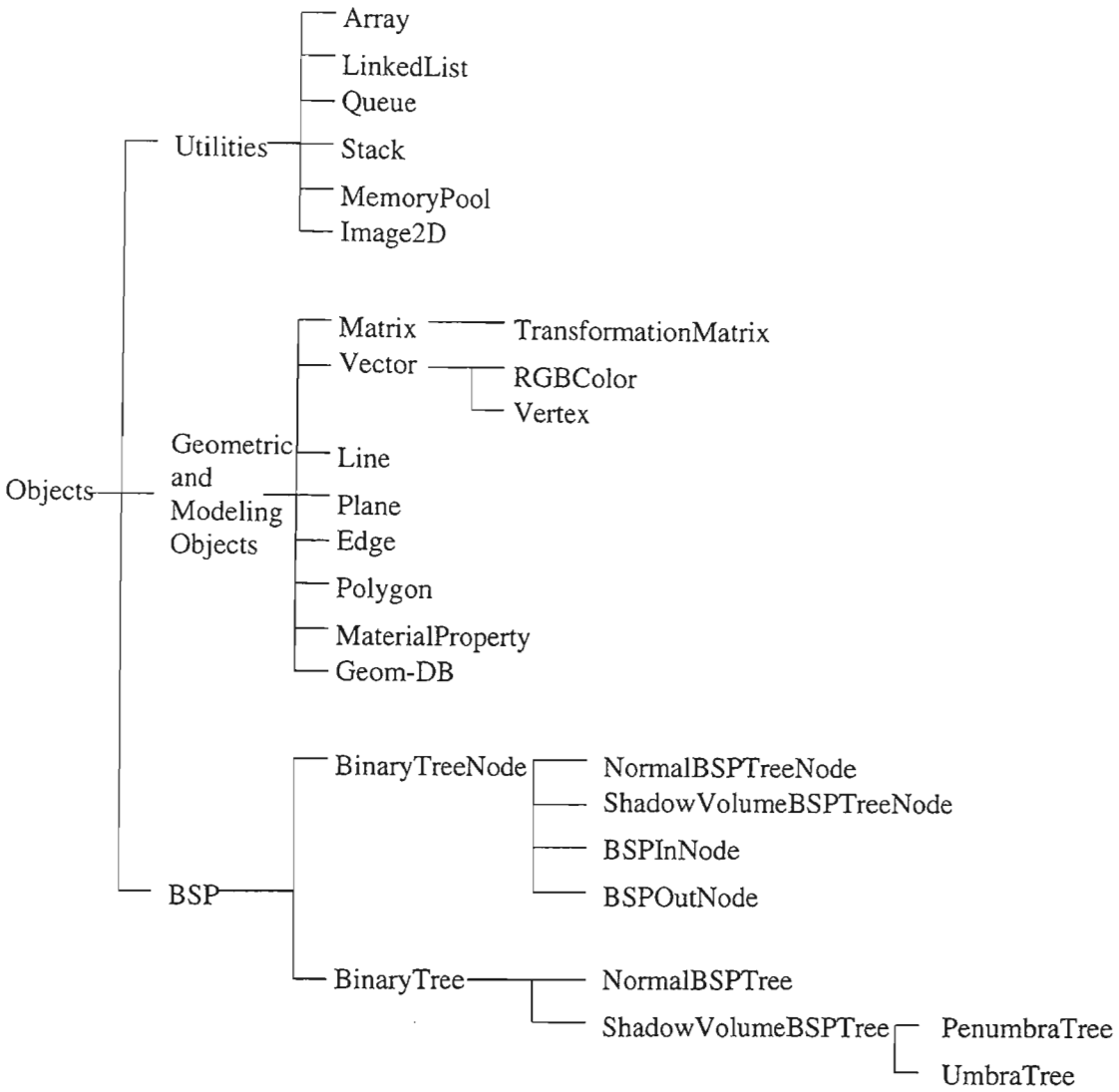


Figure 6.2. Class hierarchy of objects in the OOGP programming library.

Features of a material, e.g. color and reflectivity, are stored in object entities called *material-property*. Each *vertex* or *polygon* has a pointer pointing to one of these objects. This arrangement enables sharing of material properties between a surface and its sub-surfaces. Another important object in OOGP, called *geom-db*, is used as a database for

storing all the vertices, edges and polygons in the environment being processed.

A set of utility objects, such as *array*, *linked-list*, *stack* and *BSP-tree* are also included in OOGP. Various subclasses of BSP trees, such as *normal-bsp*, *penumbra-tree* and *umbra-tree*, are implemented and used for different purposes. These objects were designed as general and as efficient as possible because they are used very frequently in other modules.

6.2. Input and Output Modules

There are many powerful and sophisticated software packages for modeling scenes using polygons. Some of these packages are widely used in the industry already. The input module is used to translate a scene description file in one of the commonly used formats to OOGP data structures for radiosity computation. The output module is used to save the results of the computation in an output file in Open Inventor format which can be used by other software for producing pictures, animations, or interactive architectural walk-through.

Currently the input module can converts files in Open Inventor [32], Alias [2], or 3D Studio [11] formats into OOGP data structures. Open Inventor is an open standard ASCII file format used for describing 3D environments. Alias is a sophisticated 3D modeling and rendering software package which is widely used by the high performance workstations industry. 3D Studio is another 3D modeling and rendering software package. It is available on the PC platform and therefore has a very large user community.

The output module exports files in Open Inventor format which is the only format which allows color information be assigned to individual vertex. Moreover, Open

Inventor is the superset of the Virtual Reality Modeling Language (VRML) [1] which is widely used nowadays by the World Wide Web applications. Therefore, our software package can also be used by web users for creating their web contents.

6.3. The Radiosity Engine

Both algorithms proposed in previous sections are implemented inside the radiosity engine module which is built on top of OOGP. Figure 6.3 shows the components of the radiosity engine.

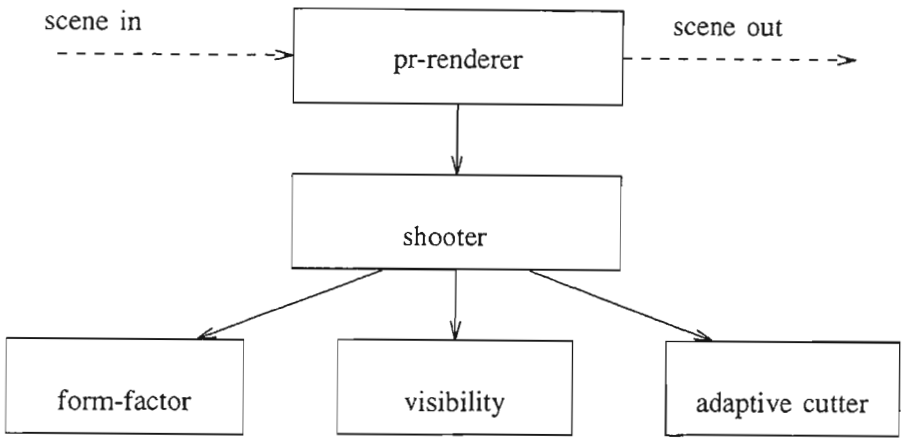


Figure 6.3. The components of the radiosity engine.

The internal data structures created by the input module are passed to an object called *pr-renderer* which performs radiosity computation using the progressive refinement approach. It proceeds in iterations. In each round, it chooses the brightest surface in the scene and passes this surface to the object called *shooter*. *Shooter* shoots out the energy of the surface chosen, and updates the radiosity of the receiving vertices in the scene. It achieves the job by making use of three other objects, *form-factor*, *visibility* and

adaptive-cutter, in the radiosity engine.

Form-factor is a functional object which uses Baum's analytical formula to compute the form-factor between a receiving vertex and a source polygon. It assumes that the source polygon is completely visible from the receiving vertex. The *visibility* object is used to determine the visibility between vertices and surfaces. It uses the improved ray tracing method with a pre-test performed using the hemicube method. The *adaptive-cutter* object uses various *BSP-tree* objects for determining the shadow boundaries on surfaces in the environment. These boundaries are used for subdividing the surfaces.

6.4. The Tuning Module

The tuning module interacts with users through a graphical panel shown in Figure 6.4. The panel contains knobs for users to specify threshold values which control the radiosity computation. The *area* threshold defines the lower bound of the area a surface can become in subdivision. This prevents the adaptive subdivision algorithm from producing too many small subpatches. The *form-factor* threshold is used to control the level of subdivision on receiving surfaces. If the form-factor between two surfaces is greater than this threshold, the algorithm will subdivide the receiving surface recursively until the threshold value is reached.

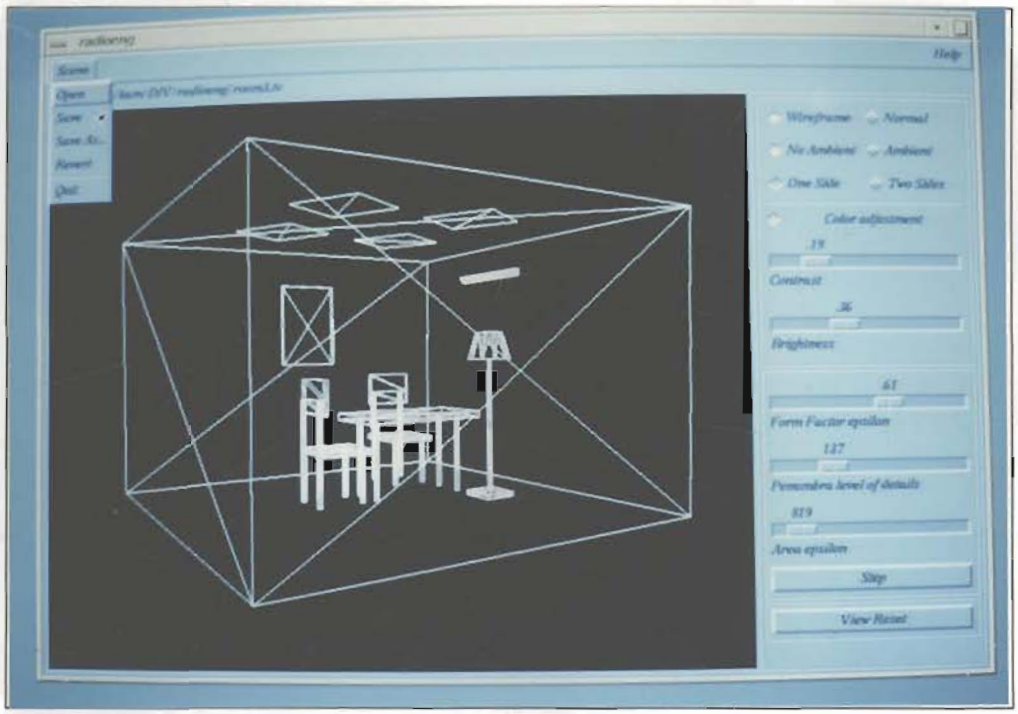


Figure 6.4. Graphical interface of the software package.

A major area of the interface panel displays the environment from a perspective view during progressive refinement. With a mouse, a user can change the viewpoint and observe the resulting scene in real time. A button can be used to toggle the display between the wireframe mode and the Gouraud shading mode.

The graphics interface is implemented using OSF/Motif programming library [21,22]. The rendering of the perspective view is implemented using OpenGL programming library [18,19]. Figure 6.5 shows the hierarchy of libraries used in various modules of our software package. It should be noticed that C++ is a multi-platform programming language and OpenGL becomes more and more popular nowadays. As a result, the software can easily be ported from one platform to another platform by just rewriting the part which uses OSF/Motif and Xlib.

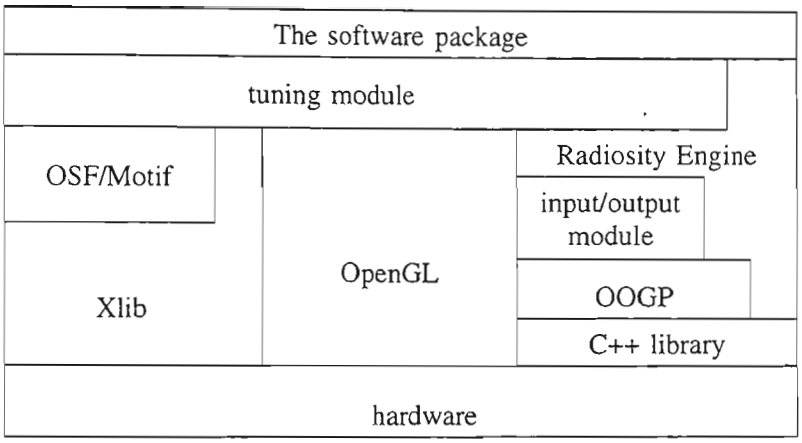


Figure 6.5. The hierarchy of libraries and modules in the software package.

CHAPTER 7

Conclusions

We have studied most of the important radiosity algorithms published in the literature. By visualizing the errors induced by these methods, a clear understanding of the advantages and shortcomings of these algorithms is obtained. These observations lead to substantial improvements in the radiosity calculation. By fully utilizing the graphics hardware and speeding up the visibility determinations using BSP tree, the improved algorithm runs four times faster than other published methods. We have also developed an adaptive subdivision algorithm which subdivides surfaces in an environment along the shadow boundaries automatically. This algorithm produces accurate shadows without any user intervention. Lastly, we have wrapped up the programs coded for radiosity calculation into a practical software package. The package imports files in one of the common scene description formats, performs the radiosity computations, and exports the result to a file in Open Inventor format. We hope that this software package will enable students in computer science or architecture to experience the radiosity method, provide computer artists an alternative means to produce photorealistic computer graphics, and allow an interior designer to visualize and market his/her design plans.

References

[1] Ames, Andrea L., David R. Nadeau, John L. Moreland, *The VRML Sourcebook*, Wiley, 1996.

[2] Anderson, Pat, Lisa Ford, Sam Hiyate, Colleen Nolan, Stephanie Noonan, *Alias Reference Guide*, Alias Research Inc., 1994.

[3] 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, July 1989, pp.325-334.

[4] Burger, Peter, Duncan Gillies, *Interactive Computer Graphics -- Functional, Procedural and Device-Level Methods*, Addison Wesley, 1989.

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

[6] Chin, Norman, Steven Feiner, "Near Real-Time Shadow Generation Using BSP Trees," *Computer Graphics (SIGGRAPH'89 Proceedings)*, Vol.23, No.3, July

1989, pp.99-106.

[7] Chin, Norman, Steven Feiner, "Fast Object-Precision Shadow Generation for Area Light Sources Using BSP Trees," *Proceedings of 1992 Symposium on Interactive 3D Graphics*, March 1992.

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

[9] Cohen, Michael F., Donald P. Greenberg, David S. Immel, Philip J. Brock, "An Efficient Radiosity Approach for Realistic Image Synthesis," *IEEE Computer Graphics and Applications*, Vol.6, No.2, March 1986, pp.26-35.

[10] Cohen, Michael F., 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, August 1988, pp.75-84.

[11] Elliott, Steven, Phillip Miller, *Inside 3D Studio, Release 4*, New Riders Publishing, 1995.

[12] Foley, J., A. van Dam, S. Feiner, and J. Hughes, *Computer Graphics: Principles and Practice*, 2nd ed., Addison Wesley, 1990.

[13] Fuchs, Henry, Zvi M. Kedem and Bruce Naylor, "On Visible Surface Generation by A Priori Tree Structures," *Computer Graphics (SIGGRAPH'80 Proceedings)*, Vol.14, No.3, July 1980, pp.124-133.

[14] Glassner, Andrew S., "Space Subdivision for Fast Ray Tracing," *IEEE Computer Graphics and Applications*, Vol.4, No.10, March 1984, pp.15-22.

[15] 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, July 1984, pp.213-222.

[16] Immel, David S., Michael F. Cohen, Donald P. Greenberg, "A Radiosity Method for Non-Diffuse Environments," *Computer Graphics (SIGGRAPH'86 Proceedings)*, Vol.20, No.4, August 1986, pp.133-142.

[17] Lischinski, Dani, Filippo Tampieri, Donald P. Greenberg, "Discontinuity Meshing for Accurate Radiosity," *IEEE Computer Graphics and Applications*, Vol.12, No.6, November 1992, pp.25-39.

[18] McLendon, Patricia, *Graphics Library Programming Guide*, Silicon Graphics, Inc., 1991.

[19] Neider, Jackie, Tom Davis, Mason Woo, *OpenGL Programming Guide*, Addison Wesley, 1993.

[20] 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, July 1985, pp.23-30.

[21] Nye, Adrian, *Xlib Programming Manual*, O'Reilly & Associates, Inc., 1990.

[22] Open Software Foundation, *OSF/Motif Programmer's Guide Release 1.1*, Prentice Hall, 1991.

[23] Paterson, Michael S., F. Frances Yao, "Binary Partitions with Applications to Hidden-Surface Removal and Solid Modelling," *Proceedings of the Fifth Annual Symposium on Computational Geometry*, 1989, pp.23-32.

[24] Shao, M.Z., Q.S. Peng, Y.D. Liang, "A New Radiosity Approach by Procedural Refinements for Realistic Image Synthesis," *Computer Graphics (SIGGRAPH'88 Proceedings)*, Vol.22, No.4, August 1988, pp.93-101.

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

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

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

[28] Stroustrup, Bjarne, *The C++ Programming Language*, 2nd ed., Addison Wesley, 1991.

[29] Thibault, William C., J. Amanatides, Bruce F. Naylor, "Merging BSP Trees Yields Polyhedral Set Operations," *Computer Graphics (SIGGRAPH'90 Proceedings)*, Vol.24, No.4, August 1990, pp.115-124.

[30] 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, July 1989, pp.315-324.

[31] 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, July 1987, pp.311-320.

[32] Wernecke, Josie, *The Inventor Mentor*, Addison Wesley, 1994.

[33] Wong, K. W., W. W. Tsang, "Three Improvements in the Ray Tracing Algorithm for Progressive Radiosity," *Tech. Report TR-93-05, Computer Science Department, The University of Hong Kong*, May 1993.

APPENDIX

A New Form-Factor Formula from a Disk to a Differential Area

Derivation of the form-factor formula from a disk to a differential area (Equation 6) :

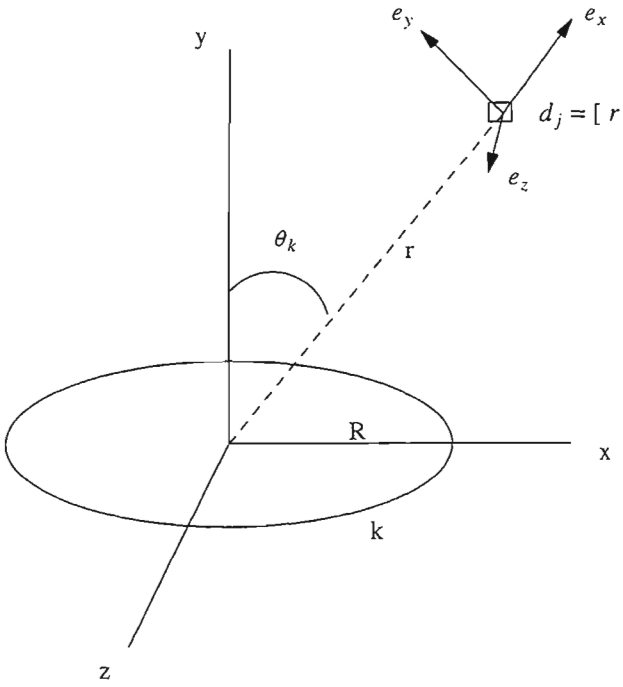


Figure A.1

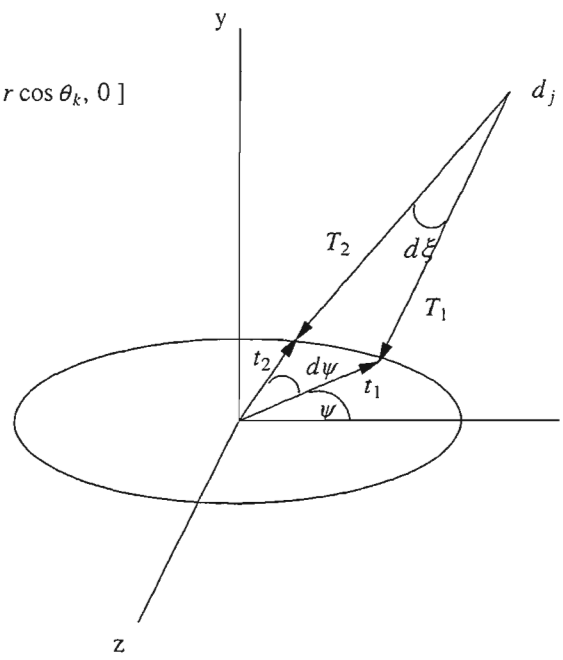


Figure A.2

WLOG, assume center of disk k is located at the origin and k lies on the x-z plane. Furthermore, assume differential area j is located at $\vec{dj} = [\ r\sin\theta_k,\ r\cos\theta_k,\ 0\ ]$ (Figure A.1).

Refer to Figure A.2:

$$\vec{t_1} = [\quad R\cos\psi,\quad 0,\quad R\sin\psi\quad]$$
$$\vec{t_2} = [\ R\cos(\psi+d\psi),\ 0,\ R\sin(\psi+d\psi)\ ]$$

$$\vec{T_1} = \vec{t_1} - \vec{dj} = [\quad R\cos\psi - r\sin\theta_k,\quad -r\cos\theta_k,\quad R\sin\psi\quad]$$
$$\vec{T_2} = \vec{t_2} - \vec{dj} = [\ R\cos(\psi+d\psi) - r\sin\theta_k, -r\cos\theta_k,\ R\sin(\psi+d\psi)\ ]$$

Let $\vec{\Gamma_{d\psi}}$ is a vector with magnitude equal to the angle $d\xi$ (in radians) and direction equal to the cross product of the vector $\vec{T_1}$ and $\vec{T_2}$.

$$\vec{\Gamma_{d\psi}} = \frac{\vec{T_1}\times\vec{T_2}}{|\vec{T_1}|\,|\vec{T_2}|\sin d\xi}\times d\xi$$

$$\approx \frac{\vec{T_1}\times\vec{T_2}}{|\vec{T_1}|\,|\vec{T_2}|} \qquad when\ d\xi\to 0$$

$$= \left[\begin{array}{c} \dfrac{-Rr\cos\theta_k[\sin(\psi+d\psi)-\sin\psi]}{\sqrt{r^2+R^2-2rR\sin\theta_k\cos\psi}\,\sqrt{r^2+R^2-2rR\sin\theta_k\cos(\psi+d\psi)}} \\ \dfrac{Rr\sin\theta_k[\sin(\psi+d\psi)-\sin\psi]-R^2\sin d\psi}{\sqrt{r^2+R^2-2rR\sin\theta_k\cos\psi}\,\sqrt{r^2+R^2-2rR\sin\theta_k\cos(\psi+d\psi)}} \\ \dfrac{Rr\cos\theta_k[\cos(\psi+d\psi)-\cos\psi]}{\sqrt{r^2+R^2-2rR\sin\theta_k\cos\psi}\,\sqrt{r^2+R^2-2rR\sin\theta_k\cos(\psi+d\psi)}} \end{array}\right]^T$$

Using the definitions above, we can write down the form-factor formula from disk k to differential area j by rewriting the Baum's form-factor formula [3] as:

$$F_{k\text{-}j} = \frac{-dA_j}{2\pi A_k} \int_{\psi} \vec{N}_j \cdot \vec{\Gamma}_{d\psi}$$

$$= \frac{-dA_j}{2\pi A_k} \times \vec{N}_j \cdot \left(\int_{\psi} \vec{\Gamma}_{d\psi} \right)$$

Before we evaluate $\int_{\psi} \vec{\Gamma}_{d\psi}$, we have made an assumption that when projecting $\int_{\psi} \vec{\Gamma}_{d\psi}$ onto the coordinate system defined by $[\ \vec{e_x}, \vec{e_y}, \vec{e_z}\ ]$ as shown in Figure A.1, the $\vec{e_y}$ and $\vec{e_z}$ component of $\int_{\psi} \vec{\Gamma}_{d\psi}$ are negligible.

$$\vec{e_x} = [\ \sin\theta_k,\ \cos\theta_k,\ 0\ ]$$
$$\vec{e_y} = [\ -cos\theta_k,\ \sin\theta_k,\ 0\ ]$$
$$\vec{e_z} = [\ \ 0,\ \ \ 0,\ \ 1\ ]$$

Project the vector $\vec{\Gamma}_{d\psi}$ onto the coordinate system $[\ \vec{e_x}, \vec{e_y}, \vec{e_z}\ ]$, we get:

$$\vec{\Gamma}_{d\psi}^{\ [e]} = \left[\begin{array}{c} \dfrac{-R^2 \cos\theta_k \sin d\psi}{\sqrt{r^2+R^2-2rR\sin\theta_k\cos\psi}\ \sqrt{r^2+R^2-2rR\sin\theta_k\cos(\psi+d\psi)}} \\ \dfrac{[Rr\cos\psi - R^2\sin\theta_k]d\psi}{\sqrt{r^2+R^2-2rR\sin\theta_k\cos\psi}\ \sqrt{r^2+R^2-2rR\sin\theta_k\cos(\psi+d\psi)}} \\ \dfrac{-Rr\cos\theta_k[\cos(\psi+d\psi)-\cos\psi]}{\sqrt{r^2+R^2-2rR\sin\theta_k\cos\psi}\ \sqrt{r^2+R^2-2rR\sin\theta_k\cos(\psi+d\psi)}} \end{array} \right]^T$$

Therefore, based on our assumption,

$$\int_{\psi} \vec{\Gamma}_{d\psi}^{\ [e]} \approx \left[\int_{\psi} \frac{-R^2 \cos\theta_k \sin d\psi}{\sqrt{r^2+R^2-2rR\sin\theta_k\cos\psi}\ \sqrt{r^2+R^2-2rR\sin\theta_k\cos(\psi+d\psi)}},\ 0,\ 0\ \right]$$

$$\approx \left[\frac{-2\pi R^2 \cos\theta_k}{r^2+R^2-2rR\sin\theta_k} \times \frac{\sqrt{r^2+R^2-2rR\sin\theta_k}}{\sqrt{r^2+R^2+2rR\sin\theta_k}},\ 0,\ 0\ \right]$$

and the form-factor from disk k to differential area j can be written as:

$$F_{k\text{-}j} = \frac{-dA_j}{2\pi A_k} \times \vec{N}_j \cdot \left(\int_{\psi} \vec{\Gamma}_{d\psi} \right)$$

$$= \frac{-dA_j}{2\pi A_k} \times \| \vec{N}_j \| \times \| \int_{\psi} \vec{\Gamma}_{d\psi} \| \times cos(\textit{angle between } \vec{N}_j \text{ and } \int_{\psi} \vec{\Gamma}_{d\psi})$$

$$\approx \frac{-dA_j}{2\pi A_k} \times \frac{-2\pi R^2 \cos\theta_k}{r^2 + R^2 - 2rR\sin\theta_k} \times \frac{\sqrt{r^2 + R^2 - 2rR\sin\theta_k}}{\sqrt{r^2 + R^2 + 2rR\sin\theta_k}} \times \cos\theta_j$$

$$= \frac{dA_j \cos\theta_k \cos\theta_j}{\pi r^2 + A_k - 2\pi rR\sin\theta_k} \times \frac{\sqrt{r^2 + R^2 - 2rR\sin\theta_k}}{\sqrt{r^2 + R^2 + 2rR\sin\theta_k}}$$