AN ANALYTIC APPROACH TO ILLUMINATION WITH AREA LIGHT SOURCES

A. T. Campbell, III and Donald S. Fussell

Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712-1188

TR-91-25 August 1991

An Analytic Approach to Illumination with Area Light Sources

A. T. Campbell, III
Donald S. Fussell
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712

ABSTRACT

Modeling the effects of area light sources has been an active area of research for many years. Most methods simplify the problem by approximating the area source with a collection of point sources. The only existing analytic methods work in screen space to compute a single image. This paper presents an object-space algorithm to model illumination from polygonal light sources, as well as point sources. The result is a collection of smooth-shaded polygonal facets that may be rendered from any viewing position. Binary Space Partitioning trees are used to compute the umbra and penumbra boundaries efficiently. Fast analytic techniques are developed for illumination calculations. Numerical optimization techniques are used to sample the shading function finely enough to find all significant illumination gradations. Illumination calculations are optimized to concentrate computational effort on parts of the scene where they are most needed.

CR Categories and Subject Descriptors: 1.3.3 [Computer Graphics]: Picture/Image Generation–Display algorithms. 1.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism.

General Terms: Algorithms.

Additional Key Words and Phrases: computational geometry, data structure, diffuse, mesh generation, optimization, penumbra, sampling, shadow.

1. INTRODUCTION

Generating photorealistic images of most scene models requires determining the illumination of surface elements by area light sources. Some of these light sources are the original emitters of the light into the environment, and others may be surface elements that reflect light onto each other. Although light source primitives currently used in generating photorealistic images are frequently polygonal areas, these are usually treated as collections of point sources.

When only diffuse emission and reflection are present, illumination does not change in a static scene as the viewing position is moved. If the illumination is precomputed, a sequence of highly realistic images from different viewpoints can be generated at interactive speeds. For most current graphics hardware, the best results toward these ends may be achieved by rendering a scene as a smooth-shaded polygonal mesh. The precomputation involves subdividing a scene into an appropriate mesh of surface elements and computing the intensities at the vertices of each element. Many mesh elements are required to sample the intensity function adequately in regions where the illumination varies rapidly. However, a large number of mesh elements slows the illumination computation and the final shading and display of the scene. To balance the requirements of realism and speed, only elements whose illumination cannot be effectively treated as constant should be subdivided. The technique presented here is effective at accomplishing this goal for either point or area sources.

This paper presents an analytic method for determining the illumination provided by area light sources of constant intensity in object space without resorting to point sampling of the sources. The method is developed in a flexible framework that can easily handle point source illumination as well. Area light sources and illuminated surfaces may be convex polygons of arbitrary shape and size. While it is presented here as a means of local penumbra illumination, this approach can also be used for global illumination.

The next section more formally presents the problem and reviews previous efforts to solve it. Section 3 describes our approach thoroughly. Implementation details are provided in Section 4. Performance results and example images are in Section 5. In Section 6 we analyze the work and suggest possible directions for further research.

2. THE SOFT SHADOW PROBLEM

Let us now describe the problem more precisely. Input consists of the scene geometry, a list of light sources, the reflectance and emittance specifications for the receivers and light sources, and an intensity tolerance, I_ϵ, to be described below.

The scene is composed of closed polyhedra. Each face of a polyhedron is a planar convex polygon. The plane of a face divides space into two subspaces with respect to that face:

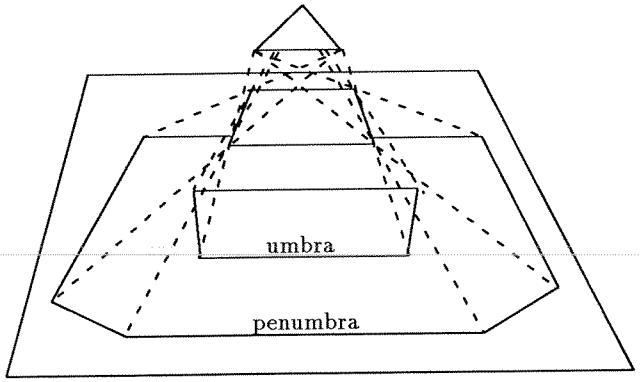


Figure 1: Soft shadows

a positive subspace, and a negative subspace. A face may be considered to be one-sided; it is visible only when viewed from its positive subspace, which is outside the polyhedron of which it is a part. The boundary of a face is represented by a list of its vertices, which are numbered V_1, V_2, $\ldots V_n$, ordered consecutively counterclockwise when viewed from the positive side.

Light sources are either points or polygonal areas. A point source is defined by a position in space. An area light source is specified as a face of any scene polyhedron.

Shading computations are performed at a finite set of wavelengths, λ_1, λ_2, $\ldots \lambda_w$. Reflection properties for each receiving surface are specified by a list of reflectance values at each wavelength of interest. To maintain physical constraints, each reflectance value is a real number in the range $[0, 1]$. Similarly, the emission of a light source must be supplied at each pertinent wavelength.

The goal is to divide the scene polygons into fragments whose intensity varies by less than I_ϵ. Intensities will be computed at all vertices of each fragment, and these polygonal fragments will then be rendered with smooth interpolation of these intensities to produce a final image.

The problem may be broken down into several parts. First, we need to compute the portions of the light sources visible from any point, P, in the scene. This information must be incorporated into a shading function to determine the intensity of P. The variation of the shading function across each surface must be characterized to determine a subdivision that allows each fragment's intensity to differ by less than I_ϵ throughout. Next, the intensities at all vertices must be computed. Finally, an image is created by rendering the polygonal fragments.

Area light sources lead to an image with soft-edged shadows. For each such light source, the surfaces in the scene may be divided into three illumination classifications: fully lit, partially lit (penumbra), and fully occluded (umbra). This is shown in Figure 1.

2.1. Shading Computations

The bulk of research in shading algorithms has been devoted to point light illumination. In this case, the light source is either totally visible or totally invisible from any point. The intensity of an occluded point is 0, and the intensity of any lit point may be computed by a simple analytic formula. The main difficulty is determining occlusions. A review of the work on this problem is beyond the scope of this paper, but the interested reader should refer to either the early survey by Crow [13] or the recent overview by Woo, Poulin, and Fournier [34].

Most existing area illumination algorithms model a source as a finite collection of point sources. Point shadow algorithms are then applied for each sample point. This has been done with the depth buffer [20], ray tracing [12] [23], and shadow volumes [5]. Unless care is taken, inaccuracies in the approximation of the illumination of an area light source may result, and aliasing may occur.

Radiosity algorithms have also been used to model area sources. These methods operate by subdividing all surfaces in a scene into patches, and then computing a geometric form factor, F_{ij}, between each pair of patches. Computing each form factor requires determining interpatch visibility. At this stage, existing methods either assume each patch is a point source [10] [4] [6] or perform visibility tests for strategically-chosen pairs of points on the surfaces in question [27] [33]. When examined solely with respect to occlusion computations, these methods differ little from the discrete methods mentioned above, and they suffer the same drawbacks.

Amanatides [1] developed an analytic approach for circular or spherical light sources. While this is a powerful technique, it works for no other light source shape.

Nishita and Nakamae [26] [27] have developed an analytic algorithm for polygonal sources. They use a shadow volume approach to divide surface polygons into lit, shadowed, and penumbra regions. Illumination calculations are performed in scanline order. For each point in penumbra, a shadow-clipping algorithm is performed to determine the unoccluded portion of the light source. Then an analytic function is evaluated to compute intensity. This approach does an excellent job of computing penumbras, but it fails to take advantage of spatial coherence in shadow clipping for penumbra points during scan conversion, and it fails to take advantage of frame to frame coherence since the illumination computations are done in image space.

2.2. Mesh Generation

While Gouraud shading hardware allows the display of scenes with pre-computed illumination, surprisingly little attention has been paid to the problem of discretizing a scene. Most commonly, the scene is subdivided in advance before

lighting computations are done, and this initial mesh is unchanged. Probably this is a side effect of the general philosophy of using fast hardware to preview the scene and then doing ray tracing for final rendering, common in commercially-available rendering packages for workstations.

Algorithms have been devised for dividing a mesh along the sharp boundaries of shadows cast by point sources. Atherton [2] and Chin [8] adapted object space hidden surface algorithms to this task. While this approach detects the most important intensity changes in a scene, it can fail to capture the true character of illumination across a fully lit area.

Mesh generation techniques for area sources have arisen out of research with radiosity algorithms. The earliest techniques [18] [10] required substantial user interaction to develop a good mesh. Cohen [11] developed an adaptive refinement approach in which a coarse mesh is shaded and the differences in intensity between the vertices of an element are used as a criterion for further subdivision. This approach often works well, but aliasing problems can result from a poorly-chosen initial mesh.

Heckbert [21] introduced a Monte Carlo ray-tracing algorithm that simulates the distribution of light from area sources. Photons originate from randomly selected points on the light sources and are distributed throughout the environment. All surfaces hit by more than some tolerance number of photons are subdivided, and the distribution is iterated until no more subdivision is warranted. While this technique captures all significant intensity changes, it leads to excessive refinement in bright areas of near-constant illumination.

Our previous work [6] extended the point shadow algorithm of Chin [8] to area illumination. A light source is approximated by a collection of point sources, and the scene is split across the sharp shadow boundaries associated with each point source. Intensities are computed at several points within each region of full illumination, and if these samples indicate a large intensity gradation, the region is subdivided. This method works well for small light sources, but as the number of point sources needed for an adequate approximation grows, subdivision within regions of penumbra is too fine.

To our knowledge, there has been no meshing algorithm to date that works with both hard and soft shadows. The methods of Cohen [11] and Heckbert [21] are limited by the quadtree data structure used for subdivision. Since sharp shadow boundaries seldom fall along axis-aligned planes, excessive refinement of the quadtree is required near a shadow boundary to approximate the shadow edge. Campbell's method [6] uses the more general BSP tree data structure, so it has more potential for extension to a wider range of light source geometries.

3. APPROACH

Our algorithm starts with an initial mesh composed of the minimal set of input polygons that fully describe the scene geometry. Mesh refinement and illumination computations are done for each light source separately, with all shading calculations and mesh subdivision finished for one source before moving on to another. During an illumination pass for an individual light source, the intensities of scene polygons are computed, and polygons exhibiting a significant intensity variation at any wavelength of interest are subdivided to keep this variation at or below I_ϵ, the user-supplied tolerance. Between steps, or after all light sources have been processed, the scene element subpolygons may be rendered to show an accurate, high-quality image of the scene at the current state of the illumination algorithm.

Receiving areas are subdivided to ensure the quality of the images created by a rendering of the mesh. Sharp shadows, cast by point light sources, introduce discontinuities in the illumination function. For the boundaries to be evident in the rendering, the mesh must be split at the boundaries. In addition, a region must be subdivided if the intensity varies significantly over the surface. Such variation occurs in regions where the visible portion of the light changes across the surface, as well as in large regions where the relative orientation of the light source changes dramatically.

Subdivision may also be introduced to improve the efficiency of the shading algorithm. For area sources, the visible fraction of the light source, f, may vary between 0 and 1. When shading any point, f must be calculated accurately, which is a time-intensive operation. Generally most of the scene either has total visibility to the light source, or none at all. The shading computations for these special cases are fast. There is a great time saving if the regions of full illumination and full occlusion can be separated from those of partial occlusion, so that the expensive shading and shadowing algorithm is only used in a restricted capacity. Frequently, intensity variations occur at or near the boundaries between regions of different visibility, so this subdivision serves a double purpose. Note that for point sources, division at shadow boundaries provides this visibility partitioning, with no region of partial occlusion.

The order in which light sources are processed is important. Point sources always introduce intensity discontinuities, but area sources do not. If we process point sources first, the scene polygons will already be somewhat subdivided when we process the area sources. In many cases, this existing subdivision will divide the fully lit or penumbra regions, with respect to a particular area source, finely enough to meet the intensity tolerance. On the other hand, if area sources are processed first, point sources will usually generate a great deal of further mesh refinement, since the planes used to split areas of homogeneous visibility classification would rarely coincide with sharp shadow boundaries exactly. Thus it is better to process all point sources, fol-

lowed by all area sources. The user may want to observe the progress of the algorithm interactively, and he or she may be willing to stop at an approximate solution before all lighting passes are finished. In this case, it is best to process light sources in order of decreasing energy, as in [9], so that the largest intensity contributions may be computed first. After all primary light sources have been processed, global illumination may proceed, if desired, by using the scene polygons as secondary light sources. In summary, we process the point sources ordered by decreasing energy, followed by the area sources in order of decreasing energy, and finally the scene polygons, also ordered by descending energy.

A chosen light source is used to define data structures representing exactly those volumes with some occlusion and those with total occlusion. These are called, respectively, the *occlusion* and *umbra* volumes. For a point source, these volumes are identical. The bounding planes of these volumes are determined by the vertices of the light source and those of the occluding polygons.

These volumes are used to partition the scene polygon into regions of distinct visibility classification, as in [26]. For point sources, receiving polygons are split across the planes of the shadow volumes into regions of full visibility (LIT) or no visibility (UMBRA). For area sources, first the occlusion volume is used to divide the scene into regions that are either totally lit or have at least some occlusion. The regions with occlusion are split across the planes of the umbra volumes to complete the classification. Since the main reason for visibility classification with area sources is efficiency, small polygons are not subdivided, but are tagged as lying in PENUMBRA. While this causes shading calculations to take a little longer for the current light pass, the number of final output polygons can be drastically reduced.

Once the visibility partitioning is done, additional subdivision may be desirable if the illumination across a lit or penumbra region varies greatly. This is determined by applying numerical optimization techniques to find the global minimum and maximum intensities. If these differ by more than the tolerance, the region is split to decrease the variation of each piece. This optimization and subdivision is applied recursively until the variation of all scene elements is below the tolerance. Finally, each of these fragments is shaded with respect to the current light source.

Now the scene elements may be rendered by smooth shading. Alternatively, the next light pass may begin. The algorithm continues until all light sources have been processed. Figure 2 summarizes the algorithm.

```
For each light source, S_i, do
    Build OCCLUSION volume for S_i
    Build UMBRA volume for S_i
    For each receiver, R_j, do
        Use OCCLUSION and UMBRA volumes to classify
            R_j into LIT, PENUMBRA, and UMBRA regions
        Subdivide LIT regions of R_j within tolerance
        Subdivide PENUMBRA regions of R_j within tolerance
        Shade fragments of R_j with S_i
    If desired, render the scene
```

Figure 2: Algorithm overview

3.1. Shadow Volumes

Receiving polygons are processed to partition them into fully lit, penumbra, and umbra regions for each successive light source. Note that for point sources, the penumbra component is empty. Division may be accomplished by building a data structure representing the volumes of all shadows cast by all scene polygons and then testing receiving polygons against these volumes. It is convenient to maintain separate structures for volumes that have some occlusion, called *occlusion* volumes, and for volumes having full occlusion, called *umbra* volumes. Clearly the occlusion volume totally encloses the umbra volume, but we need the two structures to distinguish the two types of shadow boundaries: lit/penumbra and penumbra/umbra.

In this work we must deal with three types of shadows: point shadows, penumbrae cast by area sources, and umbrae cast by area sources. In each case, we want to build a data structure representing all shadows of the same type cast by the light source. A building block approach in used. Specialized algorithms are used to build elementary regions of shadow, such as that cast by a light source and an individual polygon. Elementary regions are later merged to produce unified data structures for the entire scene.

In Sections 3.1.1 through 3.1.3, we describe the geometries of the various types of primitive shadow volumes. Section 4.2 describes the data structures we use to represent these shadows and shows how these data structures may be merged.

3.1.1. Point Shadow Volumes

Let us discuss shadows cast by point sources. These are crucial in several respects. They are needed for visibility classification. Also, they are an integral part of the shading algorithm for penumbra regions. Finally, since they are the simplest to describe, it make sense to start here.

Below is a theorem about the geometry of a shadow cast by a point source.

Theorem 1. *Suppose that we are given point light source S and convex blocking polygon B, facing S. The shadow volume cast by S and B is the volume, VOL_{pnt} formed by the intersection of the negative halfspace of B and the negative halfspaces of each plane that is defined by S and an edge of B and faces outward from B.*

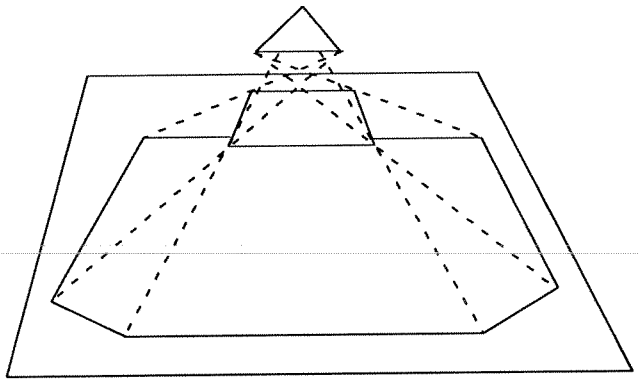


Figure 3: Occlusion volume

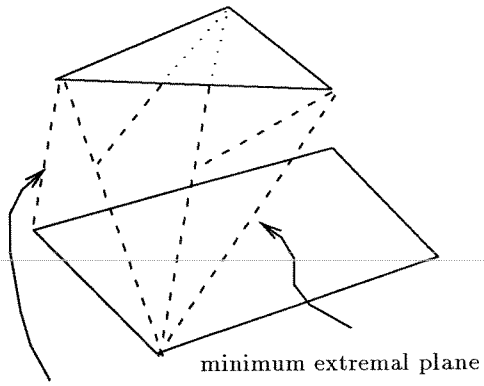


Figure 4: Choosing a plane

Proof
A point P is only in shadow if a ray of light, represented by a straight line segment from S to P, intersects B. No point between S and B is in shadow. Now let us consider the points on the negative side of B. Since B is convex, VOL_{pnt} is convex. Any line emanating from S and intersecting B will pass into VOL_{pnt} and never leave it, since each of the bounding planes is semiinfinite. Thus all points in shadow are enclosed in the volume. Similarly, no line emanating from S and not intersecting B will penetrate VOL_{pnt}. Thus VOL_{pnt} encloses the shadow exactly. □

3.1.2. Occlusion Volumes

For area light sources, an occlusion volume is the union of all points from which only a fraction $f < 1$ of a light source is visible. This is equal to the union of the point shadow volumes emanating from each point on the surface of the light source. For a convex polygonal light source and occluding surface, this is equal to the convex hull of the shadow volumes emanating from each vertex of the light source, as discussed in [26]. The computation of general three-dimensional convex hulls is notoriously difficult [16] [28]. Fortunately, the special properties of occlusion volumes allow them to be defined and computed without resorting to convex hull methods.

First, let us define a term that will help in the presentation.

Definition 1. *Let P be a point in space that is illuminated by light source S, a convex polygonal area. The paths of all light rays emanating from S and reaching P are bounded by the plane of S and all planes defined by P and each edge, (V_i, V_{i+1}) of S. The convex volume bounded by these planes is called the* illumination pyramid.

We may consider the occlusion volume to consist of all points whose illumination pyramids intersect a blocking polygon. In the immediately following text, we will develop an algorithm to compute the occlusion volume for the simplest and most common configuration of source and blocking polygon. This case is when the source and blocker face one another, and they neither cross nor touch the other's planes, as in Figure 3. In Section 4.2.2, the special cases will be considered.

Now for the simple case. The bounds of the occlusion volume must be computed. Clearly the plane of the blocker, B, is one bound. Let us consider what other planes are bounds.

Definition 2. *Let (V_i, V_{i+1}) be any edge of a blocking polygon, B. Consider all planes defined by this edge and the vertices of the light source, with the planes facing away from B. The plane that makes the smallest angle with the plane of B is called the* minimum blocker extremal plane, *and its associated vertex is called the* minimum blocker extremal vertex.

This leads to the following lemma.

Lemma 1. *No points on the positive side of a minimum blocker extremal plane may be occluded by B.*

Proof
Consider any minimum blocker extremal plane, E. Since it is defined by the vertex of S making the smallest angle with B, S must be entirely in the positive halfspace of E. Because E is tangent to the blocker and facing away from it, B is entirely in the negative halfspace of E. Now look at any point, P, on the positive side of E. The illumination pyramid for P encloses all lines from S to P. Since both P and S are all on positive side of E, the illumination pyramid is all on the positive side. Since B is on the negative side of E, it may never intersect the illumination pyramid. □

Now we have a set of planes that are known to separate unoccluded and occluded regions. But there are other bounding planes to consider.

Definition 3. *Let (V_i, V_{i+1}) be any edge of the light source, S. Consider all planes defined by this edge and the vertices of*

the blocker, with the planes facing away from B. The plane that makes the smallest angle with the plane of S is called the minimum source extremal plane, *and its associated vertex is called the* minimum source extremal vertex.

Not surprisingly, the following result holds.

Lemma 2. *No points on the positive side of a minimum source extremal plane may be occluded by B.*

Proof
In a manner similar to that of the previous proof, it may be shown that S and B are entirely on the positive and negative sides, respectively, of a minimal source extremal plane, E. Thus the illumination pyramid of any point on the positive side of E does not intersect B. □

Now we are ready to bound the occlusion volume.

Theorem 2. *Given a light source polygon S and a blocking polygon B, the occlusion volume of S and B is VOL_{occ}, the volume formed by the intersection of the negative halfspace of B, the negative halfspaces of the blocker minimum extremal planes associated with each edge of B, and the source minimum extremal planes associated with each edge of S.*

Proof
First let us consider the points outside VOL_{occ}. Clearly no blocked points may be on the positive side of B. By Lemmas 1 and 2, all points on the positive sides of the minimum extremal planes must be unoccluded. So no point outside the volume is occluded.

Consider any point, P, inside the volume. The lines connecting P to S must all intersect the plane of B. Let us find the intersection of P's illumination pyramid with B's plane. This is done by projecting S onto the plane of B. Projection is done vertex by vertex, with the projection of a vertex determined by the intersection of B's plane with the line connecting that vertex to P. S's projection is formed by connecting the projected vertices in the same order as the unprojected vertices are listed. Since S is convex, the projection is also convex. If there is no occlusion, there must be no intersection between B and the projection. Suppose there is no intersection. Then the projection must lie completely outside the boundary of B, on one side of it. This indicates that P is on the positive side of at least one minimum extremal plane. Thus there is a contradiction. So any point inside the volume is occluded. □

3.1.3. Umbra Volumes

An umbra volume is the union of all points from which none of the light source is visible. This is the intersection of the point shadow volumes emanating from each point on the surface of the light source. For a convex light source and convex occluding polygons, this is equal to the intersection of the point shadow volumes emanating from each vertex of the light source, as discussed in [26].

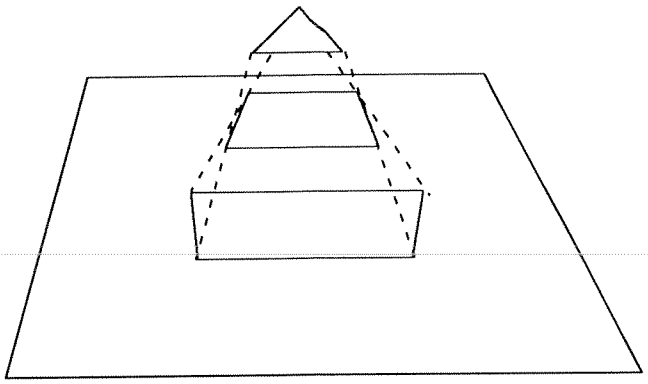


Figure 5: Umbra volume

Below is a definition that will be useful in the presentation.

Definition 4. *Let (V_i, V_{i+1}) be any edge of a blocking polygon, B. Consider all planes defined by this edge and the vertices of the light source, with the planes facing away from B. The plane that makes the largest angle with the plane of B is called the* maximum blocker extremal plane, *and its associated vertex is called the* maximum blocker extremal vertex.

This leads to the following lemma.

Lemma 3. *Any point on the positive side of the light source and the positive side of a maximum blocker extremal plane receives at least some light.*

Proof
Any point on the positive side of the light source, S, may be either on the positive or negative side of the plane of B, the blocker. Suppose P is a point on the positive side of B. Since S is also on the positive side, B can not intersect any line segments between P and S. So there is no intersection at all.

Now consider any point on the negative side of B, but on the positive side of some maximum extremal blocker plane, E. Since E makes the greatest angle with B, and S is assumed to have finite size, there must be some vertex of S that makes a smaller angle with B. Consider all points along the line segment connecting such a vertex with the extremal vertex associated with E. There must be a line passing through P and tangent to B which intersects this line segment in its interior. Unoccluded ray of light may originate from any point on this line segment that lies between the intersection point and the extremal vertex. □

Now let us completely bound the umbra.

Theorem 3. *Given a light source polygon S and a blocking polygon B, the umbra volume of S and B is the volume, VOL_{um}, formed by the intersection of the negative halfspace of B and the negative halfspaces of the blocker maximum extremal planes associated with each edge of B.*

Proof

First let us consider the points outside the VOL_{um}. Clearly no blocked points may be on the positive side of B. By Lemma 3, all points on the positive sides of the maximum extremal planes must be unoccluded. So no point outside the volume is occluded.

Consider any point, P, inside the volume. Let us compute the intersection of P's illumination pyramid with the plane of B by projecting S toward P onto B's plane, as was done in the proof of the previous theorem. P only receives light if some portion of the projection lies outside the boundary of B. This clearly can not be the case. Thus P is in umbra. □

3.2. Illumination Calculations

Once the visibility classification for a light source is complete, shading must be done. For each polygon, these calculations can be performed as follows. For umbra regions, no illumination is done. For fully lit regions, the entire light source is known to be visible from every point in the region. Thus, an analytic formula based on contour integration similar to that used in [26] can be used to determine the illumination provided by an area light source to any point on a lit surface. This formula is developed below. For regions in penumbra, it is necessary to determine the fraction of the light source visible at each point to be illuminated. Once this is done, the analytic illumination formula can them be applied using the visible portion of the light source. This is a time-consuming process, so we have developed an algorithm to limit the number of such illumination calculations required.

Illuminated regions are not, in general, regions of constant intensity. If we are to render them properly, they must be refined into regions that meet a specified tolerance for illumination variation. This effectively allows them to be later treated as constant intensity area light sources as well as providing the desired illumination sampling density. The details of mesh refinement are described in Section 4.4.

3.2.1. Point Sources

For point sources, we use the standard diffuse formula from [19]. The intensity of any point, P, is expressed by

$$I_{s-r} = I_s \rho_r (N_r \cdot L_s) \tag{1}$$

where I_s is the intensity of the light source, ρ_r is the diffuse reflectance of the receiver, N_r is the normal to the illuminated surface, and L_s is the normalized vector in the direction from P to the light source.

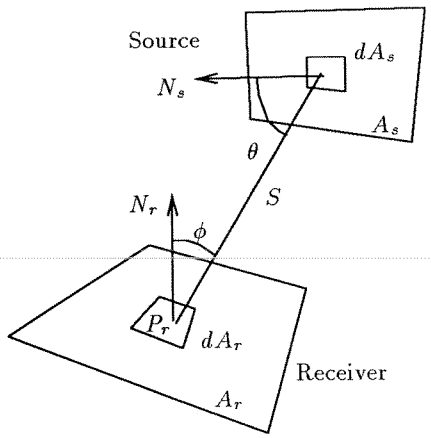


Figure 6: Geometry of illumination

3.2.2. Area Sources

Figure 6 shows the geometry of illumination. We have two polygons, A_s and A_r, which are the light source and receiver, respectively. They have unit normal vectors N_s and N_r. A_s is assumed to have uniform intensity, I_s, throughout. The diffuse reflection coefficient of the receiver is ρ_r. We want to compute the intensity of a point, P_r, on the receiver.

The relation between the emitted intensity, I_j, and the energy, E_j, of a differential area dA_j is expressed by

$$I_j = \frac{E_j}{2\pi dA_j} \tag{2}$$

Let F_{di-dj} be the fraction of energy leaving dA_i that reaches dA_j. This is known as the *geometric form factor*. Let us assume that we have a differential area, dA_r, centered at P_r. The energy received at P_r from any differential area, dA_s, on the source, and then reflected back into the environment, may be expressed by

$$E_{ds-P_r} = E_{ds-dr} = E_{ds}\rho_r F_{ds-dr} \tag{3}$$

Substituting Equation 3 into Equation 2, we get

$$I_{ds-P_r} = \frac{E_{ds-dr}}{2\pi dA_r} \tag{4}$$

$$= \frac{E_{ds}\rho_r F_{ds-dr}}{2\pi dA_r} \tag{5}$$

$$= \frac{I_s 2\pi dA_s \rho_r F_{ds-dr}}{2\pi dA_r} \tag{6}$$

$$= \frac{I_s dA_s \rho_r F_{ds-dr}}{dA_r} \tag{7}$$

From [31], we find that

$$F_{ds-dr} = \frac{dA_r \cos\phi \cos\theta}{2\pi S^2} \tag{8}$$

Substituting into Equation 7, we find that

$$I_{ds-P_r} = I_s \rho_r F_{dr-ds} \tag{9}$$

Integrating over A_s, we get

$$I_{s-P_r} = I_s \rho_r \int_{A_s} F_{dr-ds} = I_s \rho_r F_{dr-s} \qquad (10)$$

Now we need an analytic expression for F_{dr-s}. Hottel [22] provides a general expression for a form factor between a differential area and a polygon. Substituting this into Equation 10, we get the final result:

$$I_{s-P_r} = \frac{I_s \rho_r}{2\pi} \sum_{i=1}^{M_s} \arccos(V_{i-P_r} \cdot V_{(i \oplus 1)-P_r})(N_s \cdot D_i) \qquad (11)$$

where M_s is the number of vertices of A_s, V_{i-P_r} is the unit vector from the ith vertex of A_s to P_r, $\oplus$ works exactly like normal addition except that $M_s \oplus 1$ equals 1, and D_i is the unit vector in the direction of $V_{i-P_r} \times V_{(i \oplus 1)-P_r}$.

Note that our formula is slightly different from that of Nishita and Nakamae [26]. They omitted the π in the denominator. The effects of the omission are not strongly noticeable in direct illumination, since it can be corrected by scaling the source intensity. However, the effect on global illumination calculations can be dramatic.

3.3. Finding Intensity Gradations

Often it is easy to determine when the intensity variance is too high. Simply computing the vertex intensities often reveals a need to subdivide. But the maxima and minima sometimes occur on edges or in the interior of a region. We must also check for these cases. While computing and comparing vertex intensities is reasonable, the same is not true for the infinite edge and interior points. Methods are needed that are both accurate and efficient at finding the extrema for these continuous domains. Now comes the problem of finding extrema. Our method is based on numerical optimization [29]. As we shall see, the properties of each type of illumination function affect the optimization strategy chosen.

Figures 7 through 9 show the illumination function for regions fully lit by area sources. In general, there is a single maximum node and the function values decrease in all directions away from it. A maximum may occur in the interior of a polygon or along an edge. Since there is only one maximum node, local optimization techniques are applicable in finding this maximum. On the other hand, this function displays no local minima. Thus a minimum can be found solely by examination of the intensities at the vertices.

The intensity function for point illumination looks similar to that for full illumination for area sources. All of the special properties of the area function hold for the point function as well.

The illumination function for the interior of a penumbra region is difficult to characterize. As Figures 10 through

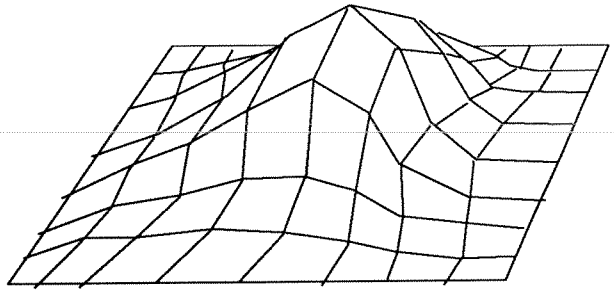


Figure 7: Fully lit, source at 0°

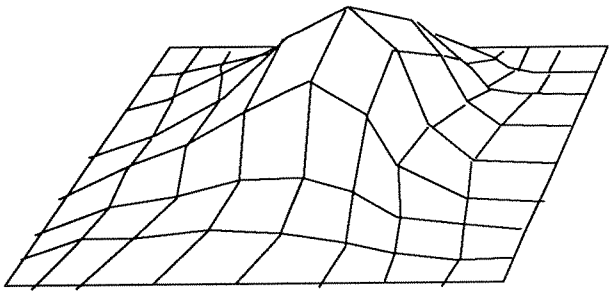


Figure 8: Fully lit, source at 30°

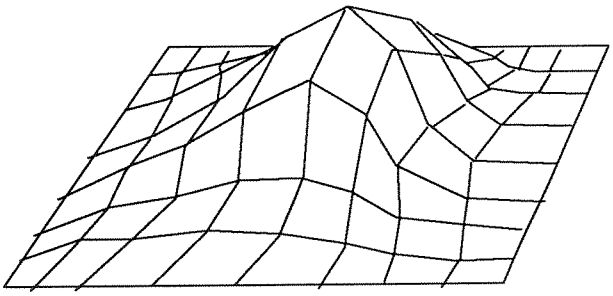


Figure 9: Fully lit, source at 60°

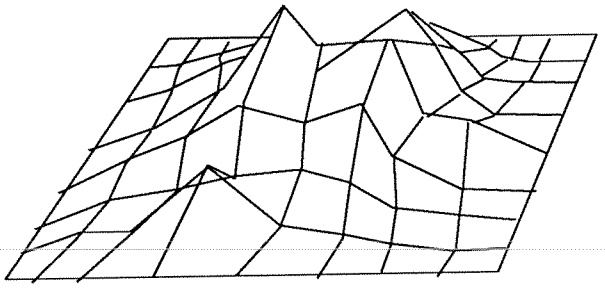


Figure 10: In penumbra, source at 0°

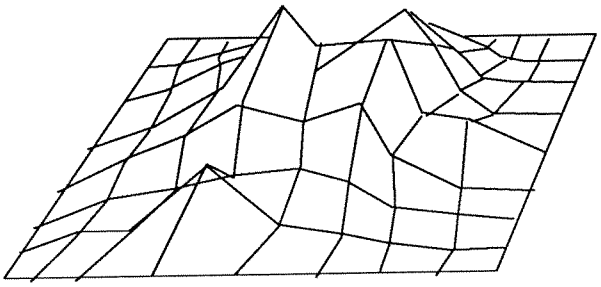


Figure 11: In penumbra, source at 30°

12 show, there may be many local minima and maxima in any given region. Thus local optimization techniques may not locate the true global extrema. More elaborate *global optimization* [14] [15] must be done.

4. IMPLEMENTATION

A detailed description of our implementation follows. As can be expected, the approach described above may be implemented in many ways.

4.1. Representation of Polygon Subdivision

During the shadowing and illumination steps, the receiver can be subjected to several stages of subdivision. Each subdivision involves splitting the surface by a plane. It is advantageous for the data structure representing this division to be hierarchical so that we may take advantage of spatial coherence in shading computations. Note than since a polygon is planar, a two-dimensional data structure is sufficient. In most existing radiosity algorithms, scene polygons are limited to rectangles and splitting is limited to planes aligned with one edge of a rectangle [10] [11] [9] [21], so this data can effectively be encoded in a quadtree [30]. Since our algorithm uses arbitrary convex polygons and generates arbitrary cutting planes, we must use a more general data structure. We use a 2-d binary-space partitioning (BSP) tree, as we did previously in [6]. Since this divides regions based on intensity and illumination criteria, let us refer to this as the *i-tree*.

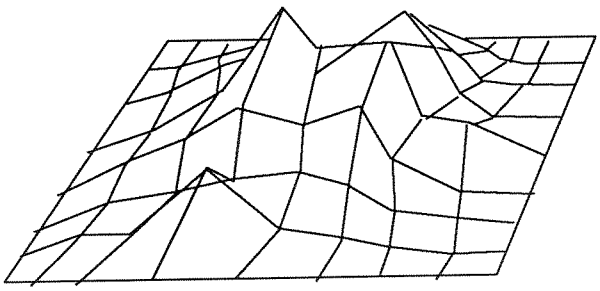


Figure 12: In penumbra, source at 60°

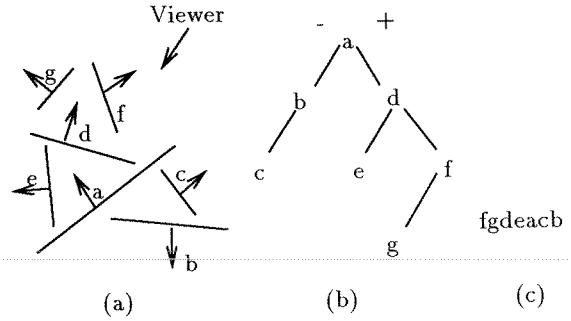


Figure 13: BSP tree

4.1.1. BSP Tree Fundamentals

A brief review of BSP trees may be helpful here. The reader may recall that a BSP tree can be used to represent a collection of n-dimensional n-gons in a volume of n-space. This is done by recursively subdividing the volume along the hyperplanes determined by the orientations of the polygons. The resultant data structure is a binary tree in which each interior node represents a partitioning hyperplane and the leaf nodes represent convex subspaces determined by the partitioning. Figure 13(a) shows a 2-dimensional example. Figure 13(b) shows the BSP tree for this scene.

As described in [17], BSP trees can be used to determine the visibility priority of a collection of polygons from any viewing position. This is achieved by a modified inorder traversal of the tree. Each node is processed recursively by inserting the coordinates of the viewing position into the planar equation of the partitioning plane at that node. The sign of the result indicates whether the viewing position is in the "front" halfspace determined by the plane, the "back" halfspace, or on the plane itself, with "front" and "back" relative to the plane normal. If the viewing position is in one of the halfspaces, the subtree representing that halfspace is processed first. If on the plane, the subtrees can be processed in any order. Once the first subtree has been processed, the polygons within the current node can be output and the other subtree processed to terminate the routine for that node. Thus the exact order of traversal is determined by the viewing position, and it is guaranteed that polygons will be listed in front to back order relative to the viewing position. Further details can be obtained in [17] or [24]. Figure 13(c) shows the output order using this algorithm on the viewing

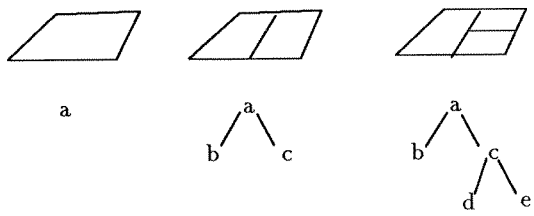


Figure 14: i-tree

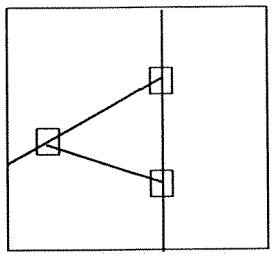


Figure 15: T vertices

position and scene shown in Figure 13(a).

4.1.2. I-Trees

Now let us describe the use of the i-tree. Its basic structure is that of a 2-D BSP tree, with the splitting hyperplanes being the lines defined by the intersections of the splitting planes with the plane of the polygon being split. An example is provided in Figure 14. At the beginning of the algorithm, each input polygon is represented by a single leaf node, representing a homogeneous region. During visibility classification, each polygon is tested for inclusion in both the umbra and penumbra volumes for that light source. The physical situation allows us to have a complete set of nonoverlapping illumination classifications: LIT, PENUMBRA, and UMBRA.

Several operations may be performed with this data structure. These include traversal, refinement, and point classification. All these important operations are described below.

4.1.3. Traversal

At the end of lighting calculations, we will need to generate a list of surface elements to be rendered. This is done by traversing the tree in any exhaustive order (inorder, preorder, or postorder) and identifying all leaf nodes. Each leaf node is attached to a global list. After all such trees have been traversed, this list is exactly what needs to be drawn.

4.1.4. Shadowing Refinement

We need to test the elements for inclusion in a shadow volume. We make use of the hierarchy for a big efficiency gain. At the root of the i-tree, we keep the boundary representation (b-rep) of the polygon it represents. With its positive child, we keep the boundary of its intersection with the positive halfspace of the dividing plane. The intersection with the negative halfspace is stored with the negative child. At each level of the tree, the appropriate recursively-defined boundary is stored. Note that the sum of the boundaries of the leaf nodes equals the whole, yet no two overlap.

During shadow testing, the boundary contour of a subtree is tested for inclusion in the appropriate shadow volume. If this region is determined not to cross a boundary of the volume, its subtrees are classified immediately as IN or OUT, and need no further examination. If a boundary of an internal node is crossed, its subtrees must be checked. If a leaf node crosses a shadow boundary and subdivision is mandated, the data structure is refined. The contour of the node is split across the divider and its fragments are associated with the two children of the newly-split node.

4.1.5. Point Classification

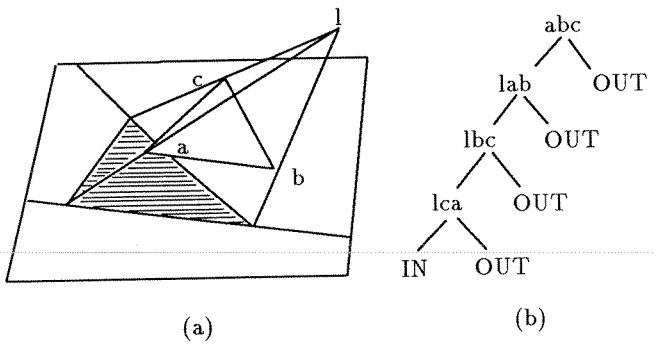
Notice that the leaves of a BSP tree are all convex subspaces. Sometimes we need to determine which of a set of points lies in or on the boundary of a leaf. Rather than exhaustively test against each leaf, we can use the hierarchy of the tree to our advantage. See [32] for another example.

Start with an i-tree and a list of points. Since each divider splits a subtree into disjoint convex regions, work may be saved by testing the points with respect to the dividing hyperplane. Substituting the coordinates of a point into the planar equation will produce a value that is less than 0, equal to 0, or greater than zero. These correspond, respectively, to points on the negative side, on the hyperplane, or on the positive side. Only points on the divider or positive side can belong to positive leaves, and the opposite is true for negative leaves. Using the tree recursively, the candidate set generated at the leaf level will all lie in the leaf as long as all points initially were in the polygon and they were not filtered out by the point classification process.

Let us give an example of its use: insertion of T vertices. Since our mesh subdivision algorithm generates an adaptive data structure of inconsistent depth, many T-vertices result. See Figure 15. Shading anomalies may result unless these vertices are added to the boundaries of all edges that contain them. This is done by first generating a list of all vertices in a polygon. Then this list is filtered down to each leaf by the i-tree. Any vertex not already in the vertex list for a leaf node must be added. Thus our problem is easily solved. Other applications of this algorithm are described later in this paper.

4.2. Implementing Shadow Volumes

An efficient data structure for shadows is the volume modeling BSP tree of [32] and [25]. These were used earlier by [8] and [6] to model sharp shadows for point sources. As



(a) (b)

Figure 16: Point shadow

```
PntShadVol(SrcPos, Blk)
root := prev := VolNode(INTERNAL)
root⇒plane := plane of Blk
for i := 1 to Blk⇒N do
    curr := VolNode(INTERNAL)
    curr⇒pos := VolNode(OUT)
    curr⇒plane := plane defined by
        SrcPos, Blk⇒V[i], Blk⇒V[i+1]
    prev⇒neg := curr
    prev := curr
prev⇒neg := VolNode(IN)
return(root)
```

Figure 17: Point shadow algorithm

shall be shown below, it is straightforward to model shadows cast by area sources as well. Special-purpose procedures produce the shadow volume cast by a source and a single convex light-blocking polygon. These volumes are then merged into a unified data structure that encloses all shadows in the scene. In Sections 4.2.1 through 4.2.3, the specialized algorithms for primitive shadow volumes are described.

4.2.1. Point Shadow Volumes

Given the geometry described in Section 3.1.1, we must now choose a data structure. BSP trees have been used to represent point shadows by [8] and [6]. The planes bounding the convex shadow volume divide space into positive and negative subspaces. From Theorem 1, the intersection of all these negative halfspaces is the shadow volume. Each leaf node of the tree has an inclusion attribute, either IN or OUT.

Consider Figure 16(a). Light source l is casting a shadow due to triangle *abc*. This leads to the data structure shown in Figure 16(b), with the triples at the internal nodes representing the defining vertices for each partitioning plane. Note that the BSP tree for the shadow cast by a convex polygon has but a single IN node.

Pseudocode for this extremely simple procedure is presented in Figure 17.

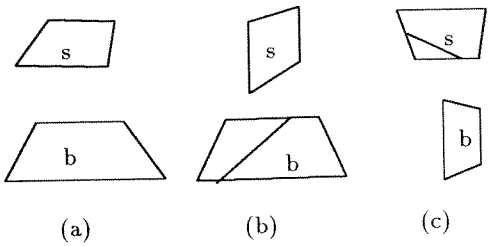


Figure 18: Source/blocker configurations

4.2.2. Occlusion Volumes

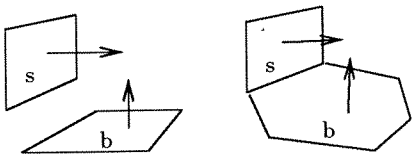
In Section 3.1.2, the geometry of occlusion volumes for the most common source/blocker configuration was described. This leads to a straightforward algorithm for computing the shadow data structure. Before we present the algorithm, let us consider the special cases. Since the light source and blocking polygon are arbitrary convex polygons, it is possible that one may cross the plane of the other, as Figure 18 shows. Since light sources only emit light from their positive faces, we need only concern ourselves with the portion of the blocker in the positive halfspace of the light source plane. The blocker is clipped against this plane, and the positive portion, if such exists, is called the *effective blocker*.

The light source may cross the plane of the blocker. If we split the light source where it crosses this plane, we have will have at most two pieces, since the light source is convex. One fragment of the light source can be in front of the blocker, and the other may lie behind it. Since our implementation assumes that all objects are closed, we may ignore backfacing polygons as potentially casting shadows, because a front face of the same object must block the same light. This backfacing test, a common optimization in hidden surface calculations, increases efficiency here as well. So we only need to be concerned with that portion of the light source, if any, in the positive halfspace of the blocker. This is called the *effective light source*.

An empty effective blocker or light source indicates that there is no shadow to compute. The BSP tree for this shadow volume, consisting of a single OUT node, is created, and the procedure is complete.

If both the effective light source and blocker polygons do exist, we may proceed with computation of the shadow volume. The source and blocker polygons are guaranteed to face each other, although they may well share edges or vertices, as Figure 19 shows. From this point on in the algorithm, we consider the effective source and blocker to *be* the source and blocker.

There is one degenerate case to consider. What if the source and blocker share one or more vertices? Unless the algorithm handles this case, some of the planes examined by the algorithm will be ill-defined, since they will only have two unique defining vertices. The answer is to omit planes if either vertex of the current edge is shared by a vertex on the other polygon. The reason is that the corresponding



(a) (b)

Figure 19: Effective sources and blockers

planes should really degenerate to the blocking plane, which is already included in the list of bounds.

Theorem 2 provides an approach to computing the volume. Pseudocode for our volume BSP tree implementation of occlusion volumes is provided in Figure 20. Notice that, since the volume is the intersection of convex halfspaces, there is only one IN node in the BSP tree.

Naylor [24] and Nishita [26] have also developed algorithms to compute the occlusion volume of a polygonal light source and a single convex blocking polygon.

4.2.3. Umbra Volumes

Now let us consider the special cases. As is the case with penumbra volumes, the light source and blocking polygon must be clipped to each other's positive sides, creating effective sources and blockers. If the effective source is not the same as the original source, then some part of the light source is visible behind the blocker. In that case, there is no umbra region.

If the source and blocker share an edge or vertex, there will be a degenerate case. Here a blocker maximum extremal plane should really be the coplanar with the blocker, but pointed in the opposite direction from it. Since the plane of the vertex is among the bounds of the umbra region, the intersection of halfspaces must be empty. So there is no umbra region.

Pseudocode for forming an umbra tree is provided in Figure 21.

Nishita [26] has also developed an algorithm for umbra volumes.

4.2.4. Merging Shadow Volumes

Once the shadow volumes for individual trees have been created, they must be merged into a data structure representing all shadows in the scene. Most early methods simply keep a linked list of the primitive shadow volumes [13] [26]. But as Chin [8] has shown, shadowing calculations can be made much faster if the merged volume data structure is better able to take advantage of scene coherence. Chin demonstrated this with a single volume BSP tree representing the union of all primitive shadows.

So once the BSP trees for individual shadows have been created, we need to merge them into a BSP tree representing

```
OcclusionVol(Src, Blk)
Src := portion of Src in Blk's + halfspace
Blk := portion of Blk in Src's + halfspace
if (Src = NIL or Blk = NIL)
    return(VolNode(OUT))
root := prev := VolNode(INTERNAL)
root⇒plane := plane of Blk
for i := 1 to Blk⇒N do
    j := minimum extremal vertex for
         edge (Blk⇒V[i], Blk⇒V[i+1])
    if (j ≠ DEGENERATE)
         curr := VolNode(INTERNAL)
         curr⇒pos := VolNode(OUT)
         curr⇒plane := plane enclosing
              Src⇒V[j], Blk⇒V[i], Blk⇒V[i+1]
         prev⇒neg := curr
         prev := curr
for i := 1 to Src⇒N do
    j := minimum extremal vertex for
         edge (Src⇒V[i], Src⇒V[i+1])
    if (j ≠ DEGENERATE)
         curr := VolNode(INTERNAL)
         curr⇒pos := VolNode(OUT)
         curr⇒plane := plane enclosing
              Blk⇒V[j], Src⇒V[i], Src⇒V[i+1]
         prev⇒neg := curr
         prev := curr
prev⇒neg := VolNode(IN)
return(root)
```

Figure 20: Occlusion volume algorithm

```
UmbraVol(Src, Blk)
EffSrc := portion of Src in Blk's + halfspace
if (EffSrc ≠ Src)
        return(VolNode(OUT))
Src := EffSrc Blk := portion of Blk in Src's + halfspace
if (Src = NIL or Blk = NIL)
        return(VolNode(OUT))
root := prev := VolNode(INTERNAL)
root⇒plane := plane of Blk
for i := 1 to Blk⇒N do
    j := maximum extremal vertex for
         edge (Blk⇒V[i], Blk⇒V[i+1])
    if (j = DEGENERATE)
         return(VolNode(OUT))
    curr := VolNode(INTERNAL)
    curr⇒pos := VolNode(OUT)
    curr⇒plane := plane enclosing
         Src⇒V[j], Blk⇒V[i], Blk⇒V[i+1]
    prev⇒neg := curr
    prev := curr
prev⇒neg := VolNode(IN)
return(root)
```

Figure 21: Umbra volume algorithm

their sum. This may be done in one of two ways. Taking account the special properties of the various types of shadows, it is possible to produce efficient specialized merging operations for each of these types. The other approach is to use a general merging algorithm. We will examine each of these in greater detail.

First let us consider the specialized approach. Chin and Feiner [8] developed a technique for merging point shadows. In their algorithm, a receiver polygon is split by the merged shadow volume of previously-processed polygons into lit and shadowed regions. This determination is made when the IN or OUT leaf nodes of the shadow volume BSP tree are reached. Lit regions are then used as the basis for additional shadow subtrees. These subtrees are attached to the main shadow tree by replacing the OUT nodes that led to the lit determination in the first place.

This works for two reasons. First, polygons are processed in sorted order away from the light source, so the plane of the blocker need not, and must not, be included in the individual volume trees. This ordering is provided by creating a three-dimensional BSP tree to represent the input polygons, and then performing a simple traversal of the tree taking into account the light source position, as was done in [8] and [6]. Second, any ray emanating from the light source and passing into shadow will stay in shadow for the rest of its length. In other words, the shadow is convex with respect to the light source. When this is the case, we may subdivide and occluding polygon and know that the sums of the shadows

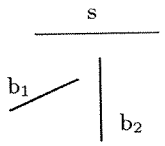


Figure 22: Ambiguous polygon ordering

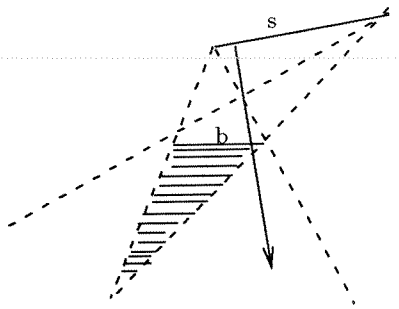


Figure 23: Penumbra and umbra shadow volumes

of its parts equals the shadow of the whole.

The Chin/Feiner merging algorithm will not work for occlusion or umbra trees. Since the light source in these cases is of finite area, there is not necessarily a possible unambiguous back-to-front ordering of scene polygons away from the source. See the example in Figure 22. From the left side of the source, polygon b_1 has priority, while from the right side of the source, b_2 should come first.

While occlusion volumes are convex with respect to the light source, this is not the case for umbra volumes. This is illustrated in Figure 23.

Early in our research, we attempted to produce specialized algorithms for merging occlusion and umbra volumes [7]. Results were disappointing. These algorithms tended to be complex, and they led to excessive subdivision in the visibility classification stage. Additionally, implementation time was substantial.

Better results were obtained when we turned to a general BSP tree merging algorithm. The method of Naylor et al [25] was used, with the set UNION operator applied to combine the trees appropriately. The Naylor algorithm is recursive, efficient, and general. Trees created in this manner have a reasonable number of nodes, and may be traversed quickly. However, this method is much slower than the Feiner/Chin method for combining point shadows.

As we will discuss in Section 5, shadow merging comprises a large portion of the run time of our algorithm. Thus it is important to choose the fastest algorithms possible. For visibility classification with area light sources, we know of no reasonable alternative to the general Naylor method. Fortunately, visibility classification is only done once per light pass, so this speed is acceptable.

However, the bulk of the shadow merging occurs during shading computations. When calculating the intensity of a point in penumbra, the unoccluded fraction of the light

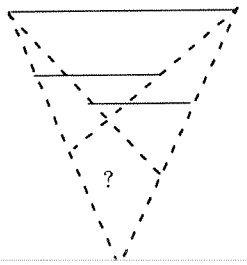


Figure 24: Difficulties with merging umbra

source must be determined. We do this by creating a point shadow volume emanating from the point to be shaded, and then we use it to clip away unseen portions of the light polygon. Details are given in Section 4.3. This computation is done many times, so it must be efficient. Here we use the Chin/Feiner algorithm, which is several times faster than Naylor's for this task.

There is an additional issue when using Naylor's general algorithm to merge occlusion and umbra trees. As described in his paper [25], the merging operation requires that additional data be stored with each node of a volume BSP tree. This data is a three-dimensional boundary representation (b-rep) of the intersection of the current node's plane with the subspace it subdivides, and must be maintained as a potentially large set of floating point triples. For small data sets ($<$ 1000 input polygons) this is not a problem, but for medium or larger inputs, a computer's memory may be exceeded. This may be handled by trading space for time. In our implementation, b-reps are never explicitly kept with each node, but are computed on the fly when needed. Experimental results have shown that the run time for the merging algorithm is increased by less than 10 percent, and the size of problems that can be successfully handled goes up by a factor of more than 10.

Now let us address a problem with creating a merged umbra volume. As shown in Figure 24, sometimes a point may not be completely occluded by a single polygon, but may be occluded by a combination of blocking polygons. Simply merging the primitive shadow volumes will not detect this case. While it is possible to design an algorithm to compute the volume in umbra due to more than one blocker, we have chosen not to do so. Such an algorithm would likely be complex and slow. Remember that the main purpose for visibility classification is speed. The incorrect umbra volume will only lead to the occasional misclassification of umbra regions as penumbra. Since the light source occlusion is computed explicitly in penumbra regions, the shading will be correct regardless.

4.2.5. Using Shadow Volumes

Now that we have created these volumes, we may use them to compute shadow boundaries. Each receiving polygon is tested in turn. At the start of shadowing, the whole scene polygon is tested with respect to the plane represented by the root of the shadow volume. If this plane is crossed, the polygon is split at the plane and the fragments are tested recursively against the appropriate subtrees. If the polygon is entirely on one side of the plane, it is then tested against the subtree on the same side. When a fragment is tested against a leaf node, it is assigned a visibility classification (lit, penumbra, or umbra) based on whether the leaf is IN or OUT. Remember that the i-tree may be used to speed up the process by providing hierarchical bounds, as discussed in Section 4.1.4.

While the above approach works, frequently more fragmentation than necessary is done. So we need a compression operation. We need to be careful not to undo mesh refinement done by a previous light pass. To this end, each node of the i-tree has an OLD/NEW boolean flag. At the beginning of an illumination pass, all nodes in each i-tree are set to be OLD. Any fragmentation done by the current pass is labeled as NEW. So the compression will only undo fragmentation labeled as NEW.

The first step in compression is to traverse the tree and see if any sibling leaf nodes have the same visibility classification. If so, they may be combined into a single node. This test and merging can be done recursively to remove all redundant nodes from the tree.

For point sources, which need sharp shadows, that is all that can be done. But consider the case of area sources. Here there is no discontinuity at shadow edges. The subdivision is merely done as an efficiency measure. We do not want to fragment too finely, since the efficiency gains would be minimal and the memory cost for the i-tree would grow unnecessarily. Note that our algorithm will always correctly shade any point classified in penumbra, so no inaccuracy is introduced by classifying a slightly larger than necessary area as penumbra.

Thus the merging algorithm is slightly modified. An absolute lower limit on polygon fragment size is set. For sibling leaf nodes of types LIT or UMBRA, the algorithm proceeds as described above. But if there are two sibling nodes where one is in PENUMBRA, the two are merged if either is below the minimum size, and the merged node is classified as PENUMBRA. This process is recursively applied. The result is a compact description of the illumination behavior on this polygon. Figure 25 provides pseudocode for this procedure.

4.3. Computing Intensities

The intensity of any fully lit point may be calculated by application of Equation 11. But for a point in penumbra, we must first find the unoccluded portion of the light source. The intensity contributions of all visible light source fragments may then be computed and summed. Point shadow volumes (Section 3.1.1) may be applied to this task. A shadow volume, emanating from the receiving point, may be built from the polygons between the source and receiver

```
CompressVisClass(Tree)
if (Tree = NIL) return
if (Tree is leaf) return
CompressVisClass(Tree⇒Neg)
CompressVisClass(Tree⇒Pos)
if ((Tree⇒Pos is leaf) and (Tree⇒Neg is leaf) and
   (Tree⇒Pos is new) and (Tree⇒Neg is new))
        if (Tree⇒Pos⇒Type = Tree⇒Neg⇒Type)
            Tree⇒Type := Tree⇒Pos⇒Type
            Destroy Tree's Children
        else if ((Tree⇒Pos⇒Type = PENUMBRA or
            Tree⇒Neg⇒Type = PENUMBRA) and
            (Tree⇒Pos⇒Area < MINSIZE or
            Tree⇒Neg⇒Area < MINSIZE))
                Tree⇒Type := PENUMBRA
                Destroy Tree's Children
```

Figure 25: Compressing visibility classification

polygons. This volume is then used to clip away occluded portions of the light source.

To build a shadow volume properly, we need to determine which polygons lie between the source and the receiver. Clearly we could test all scene polygons, but more efficiency is desirable. Potentially many points on the receiver may need their intensity computed. Our algorithm needs to take advantage of the coherence of the situation.

This may be achieved by computing a bounding volume enclosing the light source and receiving polygon. All scene polygons may be tested for inclusion in this volume, and those found to be completely outside it are removed from consideration. We use a volume bounded by the light source plane, the receiver plane, and the planes that form the convex enclosure of the two polygons. The method described above for computing the bounds of umbra volumes (Section 3.1.3) may be employed to compute these planes. Once the list of occluders is generated, all shading calculations may be done for the current receiver using this list.

Note that different regions of the same receiver may be occluded by different objects. In this case, some of the blocking list will be extraneous. Fortunately, we may prune this list even further with the help of the i-tree. Shading need only be done on subpolygons represented by leaves of the i-tree. When we compute the intensity for a point in such a region, we only want to deal those polygons that occlude the light reaching it.

This may be done as follows. First, generate a list of all occluders, using the b-rep of the entire scene polygon. As the tree is descended, the list of occluders for a parent is tested against the bounding volumes enclosing the light source and the b-reps of each of its children. While the occluders for each child are likely to overlap, there is frequently some difference. This list refinement is done recursively until the leaf nodes are reached. Refinement does not generally take much time, but it can lead to a big speedup in shading calculations. When a leaf node is reached, we may be assured that we do have a list of definite occluders.

4.4. Mesh Refinement

Now that we have an analytic formula for the intensity of any point, we know that our computed values will be accurate. But to sample all the interesting illumination effects, we must make sure that the difference between the minimum and maximum intensities of any receiver does not exceed a threshold. Additional computation must be done to find any regions that exhibit unacceptable levels of variation. These regions must be subdivided, and the resultant pieces recursively tested until all fragments are within a user-specified tolerance. Note that this is not a problem unique to area source illumination.

Let us first discuss efficiency. Remember that all shading calculations are performed on homogeneous polygonal regions represented by leaf nodes in the i-tree. Usually, the illumination function extrema in such a region lie at the vertices. If possible, we want to limit testing for edge and interior extrema to those regions that have them. This may be done by first finding all edge and interior extrema for the entire scene polygon, and then using the i-tree point classification algorithm of Section 4.1.5 to determine where these extrema belong. The problem of finding the extrema on a subpolygon is thus reduced to finding the minimum and maximum of the illumination function at a finite set of points: the vertices, plus a few others. Efficiency gains are great.

4.4.1. Fully Lit Regions

Equations 1 and 11 give the intensities of points fully lit by point and area sources. They are continuous and differentiable, so we can compute the intensity gradient. The availability of gradient information allows the use of faster, more reliable optimization techniques.

To find edge extrema, each edge represented in the i-tree must be examined. We employ Brent's method with derivatives [29], using the two bounding vertices of an edge as the initial interval. If the optimization algorithm attempts to step outside this boundary, we know that there is no local extremum on the edge, so the edge extremum must be at a vertex.

For computing interior extrema, we use the Fletcher-Powell method [29], a *quasi-Newton* method. The centroid of the polygon is used as a starting point. If any step of the algorithm moves outside the boundary, we know that there is no interior local extremum to be found. Due to the nature of the illumination function, at most one maximum may be found here.

All the extrema, if any, found for this polygon's edges and interior are gathered into a single list. This is returned to the optimization control routine.

4.4.2. Penumbra Regions

As mentioned earlier, global optimization is required in regions of penumbra. Let us first review the most common existing global optimization techniques.

The simplest approach is Pure Random Search (PRS) [14]. Here the function is evaluated at several points distributed throughout the domain. The extreme function values located are returned as the global minimum and maximum. This technique works well if the number of search points is large enough, but it is time-intensive.

Probably the most commonly used technique is the Multistart (MS) method [14]. Here, several random points in the domain are chosen as starting values for the local minimization routines. This is generally effective, and takes less computation time than PRS due to fewer needed function evaluations.

More recently, clustering algorithms [14] have been reduced to further reduce time costs of MS. The basic assumption is that points with low function values are clustered about local minima. Since searches from the same neighborhood should lead to the same extremum, the number of starting points for local optimization may be reduced. Various algorithms have been developed to generate this set of starting points, but none has been conclusively determined to be the best.

We have chosen to develop our own clustering algorithm for the task at hand. Note that in addition to finding the global extrema, we want to find *all* of the local ones.

The 2-dimensional algorithm proceeds as follows. A course rectangular grid is overlaid on the region of interest. The function is evaluated at a random point within each grid cell. Then the cells are examined to find all cells whose function values are either all greater than those of adjacent cells, or less than all of them. The sample points of these cells are then used as the starting values of the MS method. Local optimization is done with Powell's method [29]. All local extrema found by these local searches are returned.

The 1-dimensional algorithm is identical, with intervals instead of grid cells. For local optimization, Brent's method [29] is used.

4.4.3. Choosing Splitting Planes

Once it is clear that a region must be split, we need to determine where this is to be done. In our previous work [6], regions were simply bisected with axis-aligned planes perpendicular to their largest dimensions. While this proved to work, we have found that these are not the best dividing planes.

Since we want to minimize variation, it makes sense to always divide between the locations of the minimum and maximum intensities. Our new approach is to divide at a plane halfway between the two extreme points, perpendicular to the line joining them. As can be seen in the statistics and photographs of Section 5, this works well.

4.4.4. Additional Refinement

After subdivision on shadow boundaries and across areas of high contrast, no element will have significant intensity contrast. However, rendering issues may generate a need for more mesh refinement. As discussed in [3], some meshes are better suited than others for hardware-assisted Gouraud interpolation. Elements that have large aspect ratios can lead to shading anomalies. Also, when the tree representing element subdivision (in our case, the i-tree) becomes unbalanced, other artifacts may result.

These issues may be addressed straightforwardly by adding a step, after intensity-based subdivision, in which the mesh is further subdivided according to the particular rendering concerns at hand. Since these vary depending on the specific rendering hardware or software used, we will not go into further detail here. For more information, [3] should be consulted.

4.4.5. T Vertices

As mentioned in Section 4.1.5, our mesh refinement algorithm does generate T vertices. Unless these are handled properly, shading anomalies will result. A T vertex must be inserted into the vertex lists of all subpolygons whose boundary it touches. Since point shadows introduce shading discontinuities at shadow edges, it will not suffice to maintain boundaries as pointers to shared vertex records. Each boundary must have its own copy of the vertex, since the intensities of different subpolygons may be different where they meet at these vertices.

As mentioned in Section 4.2.5, all existing nodes in the i-tree are marked as "old" before a light pass. When new vertices are generated as a result of subdivision, these are stored with the parent node of the subdivision. At the end of a light pass, the only T vertices that need to be added are these new ones.

Calculation of vertex intensities is deferred to the last possible moment. First, the mesh is refined during the visibility classification stage. Then, the refined nodes are tested for further variance, at which point further subdivision may occur until the illumination variation tolerance is met. However, vertex intensities based on the current light source are not stored at this time.

For new vertices generated by the subdivision, we need to assign an intensity based on the previous light passes. Yet we do not want to repeat all the calculations of these passes.

Since all new vertices occur at the edges of preexisting subpolygons, linear interpolation along the corresponding edge yields an acceptable intensity.

After all subdivision to intensity gradation is done, we have a refined mesh showing the illumination up to the previous light pass. At this point T vertices may be processed. Each scene polygon in turn is examined for new edges introduced by the subdivision. After a complete list of these vertices is generated for a polygon, the point classification algorithm of Section 4.1.5 is used to find which vertices are incident on which element. The vertices are inserted into the vertex list of the element, with the intensity interpolated as described above.

Now it is safe to compute the new vertex intensities. The contribution of the current light source at each vertex is computed, based on its location and the visibility classification of its associated element. This contribution is then added to the vertex's current intensity to produce a new running total, based now on light passes up to and including the current one.

4.5. Mesh Compression

The contributions of several light sources can frequently wash out intensity gradations caused by a few of them. Thus it is likely that a mesh may be too refined after several light passes. To conserve space, it is good to reduce the size of the i-tree wherever possible. So we have developed a mesh compression algorithm, applicable after any light source pass.

The basic idea is the same as in Section 4.2.5. Redundant nodes should be replaced by a single one. The vertex intensities of sibling leaf nodes are examined to see if their variation exceeds the intensity tolerance. Due to the extensive work in finding variations, we know that these vertices give a true picture of the intensity profile of the elements. If the variance is within the tolerance, the nodes may be combined into their parent, with the lists of boundary vertices and vertex intensities concatenated appropriately. This procedure, recursively applied, can save a significant amount of storage in some scenes, and never increases the storage. The time consumed by this operation is minimal.

4.6. Rendering

After any light pass, or after all such passes, the scene may be rendered. Each scene polygon is traversed to obtain the leaf elements, and these must be smooth-shaded.

5. RESULTS

We have implemented our algorithm in the C programming language.

Mesh generation and illumination computation is performed on a SUN 4/260, which is rated at ten VAX 11/780 mips. Polygon scan-conversion and smooth shading are done separately on an HP 9000 series 300. All timing figures are given for the SUN machine. Rendering time on the HP is only a few seconds. Times were measured using the UNIX *time* facility.

Figure 26: Rendered boxes

Figure 27: Mesh for boxes

Figures 26 and 27 show a rendered view of a simple scene, along with its mesh. Timing data is given in Figure 28, with storage statistics in Figure 29. Nearly all of the computation time for numerical optimization is spent on evaluating the intensity functions, so we separate the function evaluations used in optimization from those used for final rendering.

6. CONCLUSIONS AND FURTHER WORK

We have presented an algorithm for analytic computation of soft shadows in object space. It operates by subdividing light-receiving polygons into homogeneous regions of three types: fully lit, penumbra, and umbra. A BSP tree data structure aids in the shadow computations. Intensities of illuminated points are then computed analytically. Techniques from numerical optimization are employed to ensure that the illumination function is sampled with enough fre-

Statistic	Figure
Name	Boxes
Input Polygons	20
Output Polygons	219
Visibility Classification (sec)	9.82
Occluding Polygon Culling (sec)	4.82
Source Occlusion (Optimization) (sec)	140.78
Shading (Optimization) (sec)	10.06
Source Occlusion (Final) (sec)	13.30
Shading (Final) (sec)	2.47
T Vertex Insertion (sec)	0.58
Total Time (sec)	181.83

Figure 28: Timing statistics

Statistic	Tree		
Name	Occlusion	Umbra	Pnt Vis (Avg)
Nodes	395	97	51
Height	17	13	10
Storage (Kbytes)	12.7	3.1	1.8

Figure 29: Storage statistics

quency.

There are several areas ripe for future work. A thorough analysis of the time and space requirements for creating and merging the various types of shadow data structures needs to be done. The optimization techniques used by our algorithm could possibly be improved by taking more advantage of the special properties of the problem. In particular, the new global optimization method introduced by this paper should be examined both to pinpoint its strengths and weaknesses, and to determine whether it has general applicability beyond the scope of the current problem.

We feel that the work presented here will be of general use to the computer graphics community. Analytic illumination techniques, along with mesh generation, have in our opinion not gotten enough attention in recent research. These are important areas that need further development.

7. ACKNOWLEDGEMENTS

We would like to thank Alan Cline for introducing us to the global optimization literature. Nelson Max made helpful observations about our visibility classification algorithm. John Howell provided references that were used in our derivation of intensity functions. Special thanks go to Philip D. Heermann of ALFA Engineering, Inc. for encouragement and advice.

REFERENCES

[1] John Amanatides. Ray tracing with cones. *Computer Graphics*, 18(3):129–136, July 1984.

[2] Peter Atherton, Kevin Weiler, and Donald P. Greenberg. Polygon shadow generation. *Computer Graphics*, 12(3):275–281, August 1978.

[3] Daniel R. Baum, Stephen Mann, Kevin P. Smith, and James R. Winget. Making radiosity usable: Automatic preprocessing and meshing techniques for the generation of accurate radiosity solutions. *Computer Graphics*, 25(3):?–?, July 1991.

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

[5] Lynne Shapiro Brotman and Norman Badler. Generating soft shadows with a depth buffer algorithm. *IEEE Computer Graphics and Applications*, 4(3):71–81, March 1984.

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

[7] A. T. Campbell, III and Donald S. Fussell. Analytic illumination with polygonal light sources. Department of Computer Sciences, Technical Report TR-91-15, University of Texas at Austin, April 1991.

[8] Norman Chin and Steven Feiner. Near real-time shadow generation using bsp trees. *Computer Graphics*, 23(3):99–106, July 1989.

[9] Michael F. Cohen, Shenchang Eric Chen, and John R. Wallace. A progressive refinement approach to fast radiosity image generation. *Computer Graphics*, 22(4), August 1988.

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

[11] Michael F. Cohen, Donald P. Greenberg, David S. Immel, and Philip J. Brock. An efficient radiosity approach for realistic image synthesis. *IEEE Computer Graphics and Applications*, 6(3):26–35, March 1986.

[12] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. *Computer Graphics*, 18(3):137–145, July 1984.

[13] Franklin C. Crow. Shadow algorithms for computer graphics. *Computer Graphics*, 11(2):242–248, August 1977.

[14] L. C. W. Dixon and G. P. Szegö, editors. *Towards Global Optimisation.* North-Holland, New York, 1975.

[15] L. C. W. Dixon and G. P. Szegö, editors. *Towards Global Optimisation 2.* North-Holland, New York, 1979.

[16] Herbert Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer-Verlag, New York, 1987.

[17] Henry Fuchs, Zvi M. Kedem, and Bruce Naylor. On visible surface generation by a priori tree structures. *Computer Graphics*, 14(3):175–181, June 1980.

[18] Cindy Goral, Kenneth E. Torrance, Donald P. Greenberg, and Benett Battaile. Modeling the interaction of light between diffuse surfaces. *Computer Graphics*, 18(3):213–222, July 1984.

[19] Henri Gouraud. Continuous shading of curved surfaces. *IEEE Transactions on Computers*, 20(6):623–628, June 1971.

[20] Paul Haeberli and Kurt Akeley. The accumulation buffer: Hardware support for high-quality rendering. *Computer Graphics*, 24(4):309–318, August 1990.

[21] Paul S. Heckbert. Adaptive radiosity textures for bidirectional ray tracing. *Computer Graphics*, 24(4):145–154, August 1990.

[22] Hoyt C. Hottel and Adel F. Sarofim. *Radiative Transfer.* McGraw Hill, New York, 1967.

[23] James T. Kajiya. The rendering equation. *Computer Graphics*, 20(4):143–150, August 1986.

[24] Bruce Naylor. *A Priori Based Techniques for Determining Visibility Priority for 3-D Scenes.* PhD thesis, University of Texas at Dallas, May 1981.

[25] Bruce Naylor, John Amanatides, and William Thibault. Merging bsp trees yields polyhedral set operations. *Computer Graphics*, 24(4):115–124, August 1990.

[26] Tomoyuki Nishita and Eihachiro Nakamae. Halftone representation of 3-d objects illuminated by area sources or polyhedron sources. In *IEEE COMPSAC*, pages 237–242, November 1983.

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

[28] Franco P. Preparata and Michael Ian Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, New York, 1985.

[29] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computation.* Cambridge University Press, New York, 1988.

[30] Hanan Samet. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, New York, 1990.

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

[32] William Thibault and Bruce Naylor. Set operations on polyhedra using binary space partitioning trees. *Computer Graphics*, 21(3):315–324, July 1987.

[33] John R. Wallace, Kells A. Elmquist, and Eric A. Haines. A ray tracing algorithm for progressive radiosity. *Computer Graphics*, 23(3):315–324, July 1989.

[34] Andrew Woo, Pierre Poulin, and Alain Fournier. A survey of shadow algorithms. *IEEE Computer Graphics and Applications*, 10(6):13–32, November 1990.

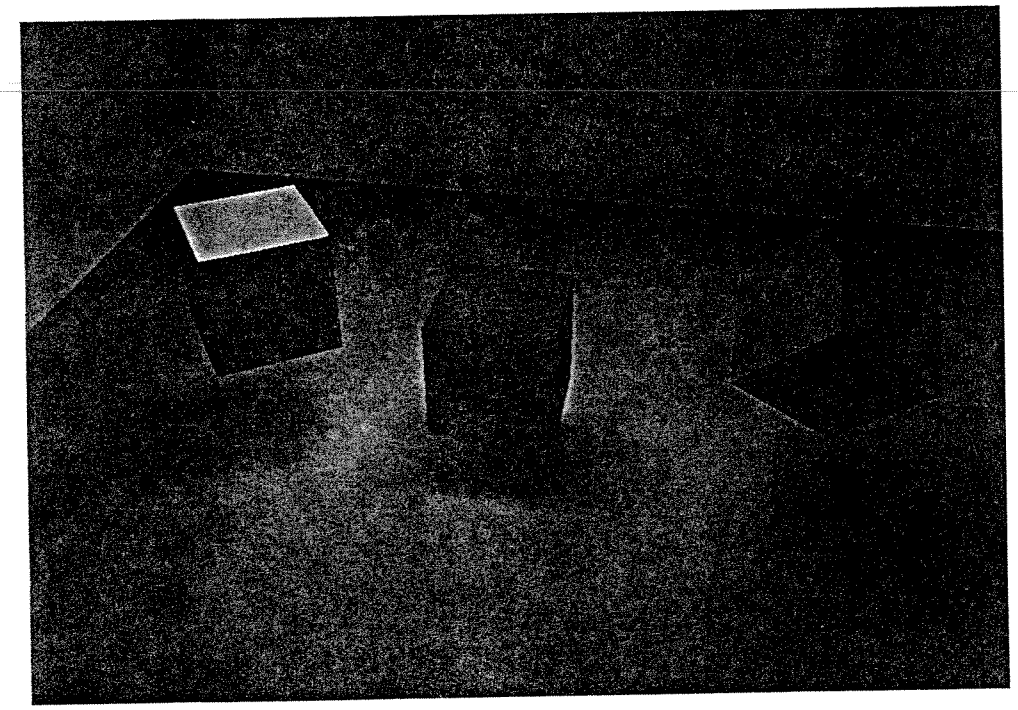


Figure 26

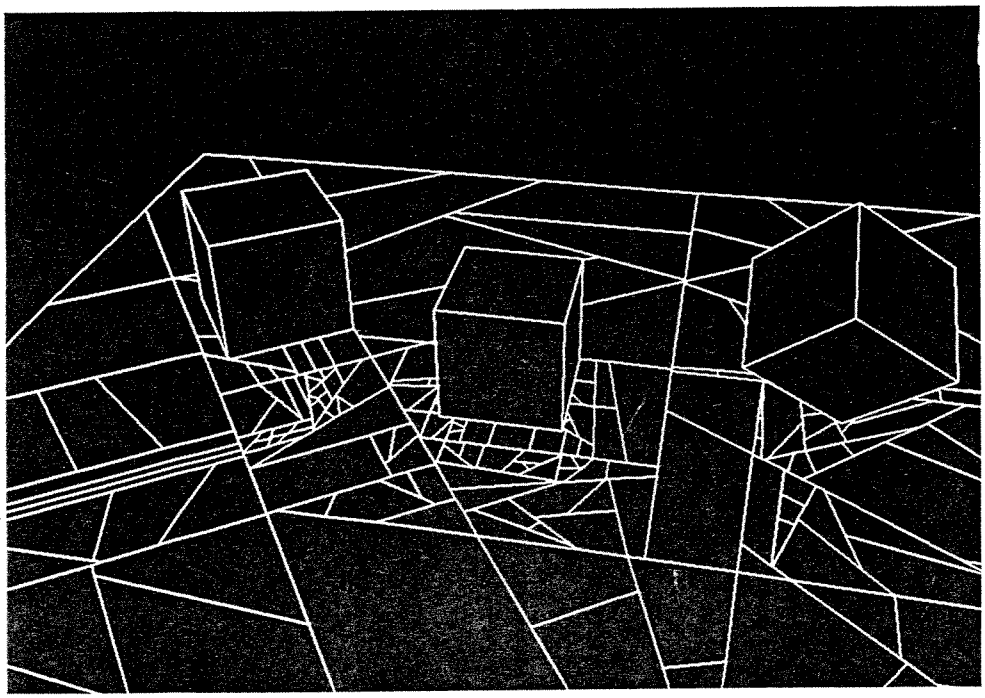


Figure 27