ANALYTIC ILLUMINATION WITH POLYGONAL LIGHT SOURCES

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

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

TR-91-15 April 1991

Analytic Illumination with Polygonal 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 finite-sized 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 method works in screen space to compute a single image. This paper presents an object-space algorithm to model illumination from polygonal light sources. The result is a collection of smooth-shaded polygonal facets which 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 in unshadowed regions 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: 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 in the environment, and others may be surface elements which reflect light onto each other. Although light source primitives currently used in generating photorealistic images are frequently polygonal areas, these are essentially always treated as collections of point 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.

When only Lambertian surface reflectances are used, illumination does not change in a static scene when the viewing position is moved. If the illumination is precomputed, a sequence of highly realistic images from different viewing positions 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 illumination precomputation involves subdividing a scene into an appropriate mesh of surface elements and computing the illumination of each element. Large numbers of mesh elements are required to adequately sample areas of rapidly varying illumination. Such areas are most often the penumbras of shadows generated by area light sources. However, a large number of mesh elements slows the illumination computation and the final shading and display of the scene. In order 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.

The following section reviews the problem of computing penumbras and describes previous efforts to solve it. Section 3 describes our basic approach. In Sections 4 through 6, the elements of our algorithm are discussed in detail. Our implementation and results are detailed in Section 7. Section 8 analyzes the work and suggests possible directions for further research.

2. PENUMBRA COMPUTATION

The basic problem is simple to describe. Given a scene description and one or more area light sources, compute the scene illumination. For either local or global illumination methods, the result is an image with soft-edged shadows. For each 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.

Determination of soft shadows involves several steps. The occlusion of the light sources must be computed. This occlusion information must be incorporated into the shading function. Finally, the shading function must be evaluated

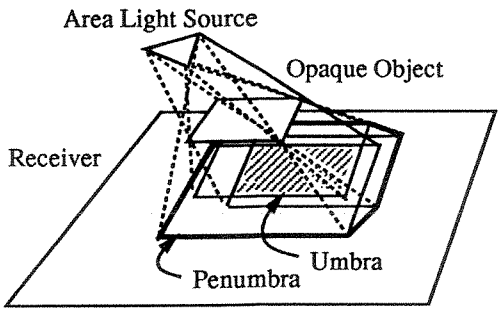

Figure 1: Soft Shadows

across each surface in the scene frequently enough to sample the intensity gradations.

The bulk of research in shadow algorithms has been devoted to point light sources. A review of this work is beyond the scope of this paper, but the interested reader should refer to either the early survey by Crow [8] or the recent overview by Woo, Poulin, and Fournier [24].

Most existing soft shadow algorithms model an area 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 [11] [12] , ray tracing [7] [15], and shadow volumes [3]. 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 generate soft shadows. 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 [6] [2] [4] or perform visibility tests for strategically-chosen pairs of points on the surfaces in question [19] [23]. When examined solely in regards to shadow computations, these methods differ little from the discrete soft shadow 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 [18] [19] 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. Our approach remedies these deficiencies in a flexible way which allows it to be employed in conjunction with view independent radiosity algorithms as well as for simple penumbra calculation.

3. ALGORITHM OVERVIEW

We assume a scene description input consisting of a list of convex polygons, some of which are designated as light sources. Light sources are processed in turn. For each light source, shadow processing is done for all receiving surfaces, resulting in a data structure which exactly represents the areas which are fully lit, in total umbra, or in penumbra with respect to the current light source. Then the illumination calculations for that light source are performed on the resulting data structure. Regions of receiving polygons in full umbra can be ignored since they receive no illumination. Regions which are fully lit by the light source can have their illumination evaluated at any point in the region using an analytic formula described below. Illumination of points in penumbra requires determination of the fraction of the light source not occluded in order to apply the analytic formula used for totally lit areas to the unoccluded light source fraction. Once the current light source is finished, the algorithm is repeated for each remaining source.

The heart of the algorithm is the first stage in which a mesh representing umbra, penumbra, and lit regions is generated. Receiving polygons are partitioned into distinct regions of visibility classification, as was done in [18]. To accomplish this, we use a generalization of the BSP tree based sharp shadow algorithm described in [5] and [4]. In this case, separate BSP trees represent the volumes with some occlusion and those with total occlusion. We refer to these, respectively, as the *shadow volume* and the *umbra volume*. Each receiving polygon is used to generate planes bounding both umbras and penumbras. These bounding planes are determined using vertices from the light source and occluding polygons as described below. We split the receiving polygon across the planes of the penumbra volume in order to divide it into regions which are either fully lit or have at least some occlusion. The areas with occlusion are then split across the planes of the umbra volume in order to complete the classification. This process continues for each receiving polygon, and when complete is repeated for each light source polygon in turn.

Once the penumbra mesh is generated, further subdivision may be desirable if the illumination across a lit or penumbra region varies by too great an amount. Since the illumination across a lit surface element is continuous and differentiable, it is possible to use numerical optimization techniques to determine whether the minimum and maximum intensities across the surface differ by more than a prescribed tolerance. If so, the receiving element is subdivided across its largest

```
For each light source, S_i, do
    For each receiver, R_j, do
        Classify R_j into LIT, PENUMBRA, and UMBRA
            regions with respect to S_i
        Illuminate R_j with S_i
```

Figure 2: Algorithm Overview

dimension. The intensity variation test is repeated until the variations for all surfaces involved are below the threshold.

Figure 2 summarizes the algorithm.

4. VISIBILITY CLASSIFICATION

Receiving polygons are processed to partition them into fully lit, penumbra, and umbra regions for each light source in turn. This may be accomplished by building a data structure which represents all shadows cast by all scene polygons, and then testing each receiving polygon against the merged shadow volumes. Separate data structures are maintained for shadow and umbra volumes. The volume modeling BSP trees of [22] and [17] can be used for this purpose. Separate trees are maintained for shadow and umbra volumes. Each leaf node of these trees has an IN/OUT attribute, indicating that the associated volume of space is inside or outside the volume. Trees representing the shadow and umbra volumes for each polygon are produced as described below, and the UNION of these volumes is performed to generate merged shadow volume trees for the umbra and shadow volumes. These trees represent all shadows cast from a single source. Receiving polygons are split where they cross the boundaries of these merged volumes to determine the boundaries of regions which are in umbra or penumbra with respect to a light source.

Efficiency gains may be achieved if the receiving polygons are processed in front-to-back order away from the light source. Each polygon may then be tested only against the merged shadow and penumbra volumes generated by the source polygon and closer polygons. For area sources, a unique ordering may not always be possible. In the case of moderately sized light sources, we can use this optimization by subdividing the source into fragments which each allow an ordering. After this subdivision has taken place, separate visibility classification passes are performed for each each fragment. The result is a mesh which classifies the scene polygons as to visibility with respect to the entire current light source.

The visibility algorithm proceeds as follows. All receiving polygons are initialized to fully lit with respect to the current light source. Then the source is fragmented as needed to allow ordering. For the visibility pass of the first fragment, lit regions are first tested against the penumbra volume. Areas

```
Initialize receivers as LIT with respect to S
Subdivide S to allow front to back ordering
For each fragment, F_i, do
    Sort receivers front to back
    Initialize shadow volumes
    For each receiver, R_j, do
        Clip LIT fragments of R_j against penumbra volume
        If i = 1 then Clip PENUMBRA fragments of R_j
            against umbra volume
        Else Clip UMBRA fragments of R_j against umbra volume
        Update shadow volumes
    Destroy shadow volumes
```

Figure 3: Visibility Algorithm

determined to lie within penumbra are then tested against the umbra volume.

Regions classified to be in penumbra in any pass do not need further visibility classification. However, we may not classify a region as fully lit or occluded until all fragment passes are complete. Regions classified as fully lit by previous passes must still be tested against penumbra volumes at each stage to detect possible occlusion. Similarly, a region must lie within the umbra volumes generated by each pass in order to be classified as fully occluded.

Pseudocode for this procedure is provided in Figure 3.

4.1. Ordering Receiving Polygons

If receiving surfaces are processed in front-to-back order from the light source, then during shadow computation receivers are shadowed only by surfaces preceding them in the list. For point sources, if a Binary Space Partitioning (BSP) tree is created to represent the input polygons, then a simple traversal of the tree taking into account the light source position can provide this ordering, as was done in [5] and [4].

The reader may recall that a BSP tree can be used to represent a collection of polygons in a volume of space by recursively subdividing the volume along planes determined by the orientations of the polygons within the volume. Polygons may be chosen in any order to determine these partitioning planes, with the most desirable order being one which results in the fewest polygons being split along plane boundaries as the process proceeds. The resulting data structure is a binary tree, in which each interior node represents a partitioning plane and its defining polygon, along with any other coplanar polygons that may exist in the input database, and the leaf nodes represent convex volumes of space determined by the partitioning. Figure 4(a) shows a two dimensional example, in which line segments represent polygons. Figure 4(b) shows the BSP tree for this scene.

As described in [9], BSP trees can be used to determine

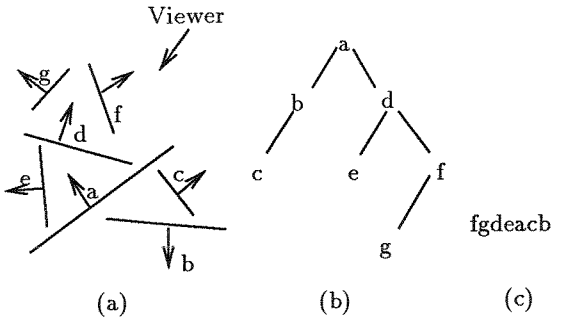

Figure 4: BSP Tree

the visibility priority of a collection of polygons from any viewing position. This is achieved by an *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 processing 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 this order will output polygons in front to back order relative to the viewing position. Further details can be obtained in [9] or [16]. Figure 4(c) shows the output order using this algorithm on the viewing position and scene shown in Figure 4(a). For a point light source, a front-to-back ordering may be generated by using the light source position as the viewpoint.

Now let us consider an area source. If each point of the source results in the same traversal sequence of the BSP tree, we know that an unambiguous sort exists, namely the order generated. As we can see in Figure 4, the ordering will change only if the light source crosses the plane of an internal node, which results in a different order of visiting its subtrees. Thus the light source must be partitioned at its intersections with each of the planes corresponding to internal nodes. For each resulting light source fragment, a front-to-back ordering of the receiving polygons is determined by traversing the tree with any point of the fragment as the viewing position.

Each fragment is treated as a separate light source for the remainder of the visibility classification algorithm. For most scenes, the fragmentation level is small.

4.2. Representation of Polygon Subdivision

During the shadowing and illumination steps, the receiver can be subjected to several stages of subdivision. Since each subdivision involves splitting the surface by a plane, the partitioning of a receiver may be represented by a two-dimensional BSP tree, as in [4]. Let us refer to this as the *partitioning tree.*

For each interior node of the partitioning tree, we keep a 3-D boundary representation of the contour of the input polygon fragment for each of its halfplanes. For each leaf node, the boundary representation of the polygon fragment it represents is maintained. This contour is tested against the boundary planes of the merged shadow and umbra volumes. If the boundary of an internal node lies entirely on one side of such a plane, then none its subtrees needs to to be examined. If an internal node crosses a plane, its subtrees must be checked against this plane. If a leaf node crosses a plane, its boundary is split across that plane and the fragments are then associated with two new nodes of the trees, which become the children of the original node.

At the start of visibility classification, all nodes of every polygon are marked as fully lit with respect to the current polygon. Each polygon is then tested for inclusion in both the shadow and umbra volumes for that light source. In general, volume BSP trees classify completely unrelated attributes [22], so membership in multiple trees would need to be represented by a complicated data structure. However, in this case the physical situation allows us to use a binary tree structure, with three different inclusion categories: $\neg$ SHADOW $\cap$ $\neg$ UMBRA $\Rightarrow$ *fully lit*, SHADOW $\cap$ $\neg$ UMBRA $\Rightarrow$ *penumbra*, and SHADOW $\cap$ UMBRA $\Rightarrow$ *umbra*. The fourth case, $\neg$ SHADOW $\cap$ UMBRA, never happens.

4.3. Point Shadow Volume

Let us begin with a discussion of shadows cast by point sources. While point shadows are not used in our algorithm for visibility determination, they are needed for shading calculations in penumbra regions. This application will be discussed below.

BSP trees have been used to represent point shadows by [5] and [4]. Planes formed by the light source and the edges of the receivers subdivide space into lit and shadowed intervals. When we process a new polygon, it is split along the planes until each fragment is determined to be in a lit or unlit region. Then the shadow volumes formed by the lit portions are added to the merged shadow volume. The insertion process can be done by representing the new shadow volumes as a BSP tree and performing set UNION operations between them and the merged shadow volume tree as defined in [17], although it may be somewhat more efficient to use the specialized procedure shown in Figure 5, which is adapted from [5].

The simplicity of this insertion procedure is due to the fact that any ray emanating from a point light source and passing into a shadow region will remain in shadow for the rest of its length, i.e. the shadow volume is convex with respect to the light source. When this is the case, we may subdivide an occluding polygon and know that the union

```
MergeVol(Vol, Src, Rcv)
If Vol.Type = IN Return(Vol)
If Vol.Type = OUT
  Vol := BuildVol(Src, Rcv)
Else
  Clip Rcv against Vol.Plane, yielding Pos and Neg fragments
  If Pos ≠ NIL then Vol.Pos := MergeVol(Vol.Pos, Src, Pos)
  If Neg ≠ NIL then Vol.Neg := MergeVol(Vol.Neg, Src, Neg)
Return(Vol)
```

Figure 5: Merging Shadow Volumes

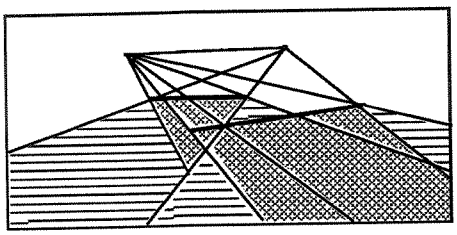

Figure 6: Penumbra and Umbra Shadow Volumes

of the shadow volumes of its fragments equals the shadow volume of the whole. This is not true in general, as shown in Figure 6.

4.4. Polygon Shadow Volumes

For area light sources, a shadow 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 [18].

The computation of general three-dimensional convex hulls is notoriously difficult. Fortunately, the special properties of polygonal shadow volumes make an efficient special-purpose algorithm possible. The process proceeds as follows. First, clip the source polygon across the plane of the occluding polygon. For the purpose of building penumbra volumes, the portion of the light on the front halfspace of the occluder is the effective light source.

Now consider all planes defined by adjacent pairs of vertices on the occluding polygon and any vertex of the light source. For each edge of the occluding polygon, the plane defined by that edge and the light source vertex with least angle above the plane of the occluding polygon forms a boundary of the shadow volume. Call this plane and its associated light source vertex the *extremal plane* and *extremal vertex* for that edge. All planes formed by other light source vertices and this edge of the occluding polygon will fall within the shadow volume, as shown in Figure 1. Thus a first approximation to the desired shadow volume is obtained by finding the extremal plane for each edge of the occluding polygon, and taking the intersection of the shadowed halfspaces formed by these planes.

This does not exactly form the shadow volume, although it does contain it, as shown in Figure 1. The erroneously included fully lit regions can be removed by computing an additional set of bounding planes. These are formed by each vertex of the occluding polygon and the two extremal vertices of the light source for the edges containing that occluding vertex. These planes again each form a shadowed and unshadowed halfspace, and these are intersected with the shadow volume formed by the extremal planes. The result is the desired convex volume, which forms the shadow volume, as illustrated in Figure 1. Note that this corresponds to the construction of the *obstruction polyhedron* defined in [16].

There are a few special cases for this latter type of plane. If the source vertices associated with both edges bounding a vertex of the occluding polygon are the same, no plane is defined. In this case, no plane containing this vertex is needed.

Several source vertices may qualify as the extremal vertices associated with one or both of the edges enclosing vertex B_i of the blocking polygon. In this case, the appropriate vertices are the pair, S_j and S_k, which form the greatest angle $\angle S_j B_i S_k$. If the vertices of each source and receiving polygon are ordered so that they appear in counterclockwise order from their "front" sides, this determination is simple. Suppose that we want to find the plane passing through blocker vertex B_i. We start with a pair of extremal vertices for the edges (B_{i-1}, B_i) and (B_i, B_{i+1}). If we travel counterclockwise about the source from the extremal vertex for (B_{i-1}, B_i), and counterclockwise from the extremal vertex for (B_i, B_{i+1}), until we encounter the last extremal vertices for their respective edge, these "outermost" vertices are the ones to use.

Pseudocode for finding penumbra planes is provided in Figure 7.

Intersections may be performed using the general set intersection operations on BSP trees described in [17]. As we can see in Figure 6, area shadow volumes are convex with respect to each point of the light source. Thus the same technique for merging these shadow volumes employed for point shadow volumes may be substituted for general set intersection operations on BSP trees.

4.5. Polygon Umbra Volume

An umbra volume is the union of all points from which none of the light source is visible. This is the intersection of the

```
ExtremeVert(Edge, Src, Rec, MoreExtreme)
best := 1
angbest := Angle(Plane(Edge, S_best), Rec)
For each light source vertex, S_j, do
    ang := Angle(Plane(Edge, S_j), Rec)
    If MoreExtreme(ang, angbest)
      best := j
      angbest := ang
Return(best)

PenumbraPlanes(Src, Rcv)
Initialize planelist to empty
last := ExtremeVert(Edge(R_n, R_1), Src, Rec, '<')
For each receiver vertex, R_i, do
    j := ExtremeVert(Edge(R_i, R_i+1), Src, Rec, '<')
    Add plane passing through R_i, R_i+1, and S_j
    While S_last is extremal
          last := last - 1
    While S_j is extremal
          j := j + 1
    If last ≠ j
          Add plane passing through S_last, R_i, and S_j
    last := j
Return(planelist)

UmbraPlanes(Src, Rcv)
Initialize planelist to empty
For each receiver vertex, R_i, do
    j := ExtremeVert(Edge(R_i, R_i+1), Src, Rec, '>')
    Add plane passing through R_i, R_i+1, and S_j
Return(planelist)
```

Figure 7: Building Shadow Volumes

point shadow volumes emanating from each point on the surface of the light source. For convex light source and occluding polygons, this is equal to the intersection of the point shadow volumes emanating from each vertex of the light source, as discussed in [18]. Note that if the light source crosses the plane of the occluder, the umbra volume is guaranteed to be empty and need not be computed.

Umbra volumes can be created in a manner analogous to the creation of shadow volumes above. Instead of those planes which, defined by the occluding edge and a light source vertex, form the least angle with the occluding plane, the extremal planes for umbra volumes are those which form the greatest angle with the plane of the occluder. Using these planes in place of those used for penumbra volumes, umbra volumes can be created with the same method. Merging of umbra regions can be performed using the general set UNION operation on BSP trees defined in [17]. Since umbra regions are not convex with respect to all points of the light source, as illustrated in Figure 6, the special case procedure used for merging point shadows and penumbra shadows will not work for umbra volumes.

Merging shadow volume trees created for distinct occluding polygons using generalized set operations can result in very large BSP trees for complex scenes. We have opted to maintain the merged volume as a simple linked list of all shadowing polygons and their umbra volumes, which are individually represented by BSP trees. This approach controls storage requirements while maintaining the simplicity of the implementation.

5. ILLUMINATION CALCULATIONS

Once the visibility classification procedure for each light source is complete, illumination calculations are performed. At this stage, each input polygon contains a two-dimensional BSP tree representing its partitioning into lit, penumbra and umbra regions. For each polygon, illumination 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 [18] 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. Fully lit regions are not, in general, regions of constant illumination. If we are to sample them properly, we must refine them until they 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. 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 the most time consuming portion of the illumination computation. The details of

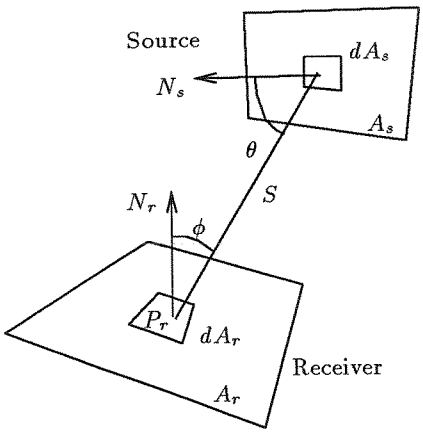

Figure 8: Geometry of Illumination

mesh refinement and penumbra illumination are described in the next section.

In the case of many light sources, or for global illumination calculations, shading calculations should be performed for each light source after visibility classification. In the case of a small number of light sources, there are advantages to deferring illumination computations until all shadow boundaries from all light sources have been computed. Since the illumination phase finely divides areas of high contrast, the final size of the output mesh is reduced if these areas with fine subdivision are not subdivided further by shadow computations. Since the 3 visibility classes for each light source may be represented with 2 bits, the storage requirements are not great if we simply keep separate visibility data for each light source within the partitioning trees of scene polygons.

Figure 8 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

$$I_j = \frac{E_j}{2\pi dA_j} \tag{1}$$

Let F_{di-dj} be the fraction of energy leaving dA_i which 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

$$E_{ds-P_r} = E_{ds-dr} = E_{ds}\rho_r F_{ds-dr} \tag{2}$$

Substituting Equation 2 into Equation 1, we get

$$I_{ds-P_r} = \frac{E_{ds-dr}}{2\pi dA_r} \tag{3}$$

$$= \frac{E_{ds}\rho_r F_{ds-dr}}{2\pi dA_r} \tag{4}$$

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

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

From [21], we find that

$$F_{ds-dr} = \frac{dA_r \cos\phi \cos\theta}{2\pi S^2} \tag{7}$$

Substituting into Equation 6, we find that

$$I_{ds-P_r} = I_s \rho_r F_{dr-ds} \tag{8}$$

Integrating over A_s, we get

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

Now we need an analytic expression for F_{dr-s}. Hottel [14] provides a general expression for a form factor between a differential area and a polygon. Substituting this into Equation 9, we get the final result:

$$I_{s-P_r} = \frac{I_s \rho_r}{2\pi} \sum_{i=1}^{M_s} \arccos(V_{i-P_r} \cdot V_{(i\oplus 1)-P_r})(N_s \cdot D_i) \tag{10}$$

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 $V_{i-P_r} \times V_{(i\oplus 1)-P_r}$.

6. 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 in order 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 which exhibit unacceptable levels of variation. These regions must be sudivided, 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.

In many cases it is easy to determine when the intensity variance is too high. Simply computing the vertex intensities often reveals a need to subdivide. But we need to be sure when subdivision is *not* needed. Not only the vertices must be examined, but also the edges and interior. Since the number of vertices is finite, it is reasonable to compute the intensities at each. But the edges and interior have an infinite number of points.

6.1. Fully Lit Regions

Equation 10 gives the intensity of any fully lit point. It is continuous and differentiable, so we can compute the intensity gradient. Computing the gradient gives us several advantages.

First, it can give a quick early termination condition for variation testing. Since the illumination function is a "nice" function, we only need to search for a local minimum or maximum within an edge or interior if the gradients at the boundary all point inward or outward. If the gradients do not point consistently, the extremum is on the boundary.

We begin by computing the intensities and intensity gradients at the receiving polygon's vertices. If the tolerance is exceeded, we subdivide. If the tolerance is not exceeded but the gradients do not consistently point in or out, we know not to subdivide. The process is repeated for edges. At every vertex examined in edge minimization, the gradient is computed. If any two gradients do not point consistently, subdivision is not needed.

If additional testing must be done, the availability of gradient information allows the use of faster, more reliable optimization techniques. For edge optimization, we employ Brent's method with derivatives [20], using the two bounding vertices as the initial interval. If the optimization algorithm attempts to make a step outside the boundary, we know that there is no local minimum on the edge, so the edge minimum must be at one of the vertices.

For computing interior extrema, we use the Fletcher-Powell method [20], one of the so-called *quasi-Newton* methods. The centroid of the polygon of interest is used as a starting point. If any step of the algorithm moves outside the boundary, we know that there is no interior local minimum to be found.

Note that this refinement suffices only for local illumination. Global illumination functions are not so simple to treat.

6.2. Penumbra Regions

To compute the intensity of a point in penumbra, we must first find the unoccluded portion of the light source. Then Equation 10 may be applied for each visible light source fragment.

Point shadow volumes (Section 4.3) 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 polygons. This volume is then used to clip away occluded portions of the light source.

This task can be particularly easy if during visibility classification, penumbra boundaries are maintained separately for each light source, and these boundaries in the 2-D BSP tree are tagged with the occluding polygon edge that produced them. In this case the appropriate shadow volume can be quickly assembled from the occluding edges bounding the region in which the point being illuminated is located. Since we have chosen not to maintain this information in order to conserve storage, our current implementation employs a different method.

To build the shadow volume properly, we need a list of intervening polygons, as well as an ordering away from the receiver. A back-to-front ordering of all scene polygons may be generated by a traversal of the scene BSP tree, using the receiver point as the viewpoint. Clearly the occluding polygons appear in this list in the proper order. But we need to determine the subset of this list which actually lies between the light source and receiver.

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 which form the convex enclosure of the two polygons. The method described above for computing the bounds of umbra volumes (Section 4.5) may be employed to compute these planes.

The illumination function for the interior of a penumbra region is not well-behaved even for local illumination. The optimization methods for lit regions cannot be used here. However, we still desire to avoid the very expensive exact illumination calculations described above for every pixel on the screen. Currently we generate a uniformly spaced set of sample locations in the penumbra region, and these are checked to see if they exceed the tolerance. If so, the region is split along its long axis, and each side is sampled. This occurs recursively until the tolerance is not exceeded.

6.3. T-Vertices

After the mesh has been refined and vertex intensities computed, an image can be displayed by Gouraud shading the receiving elements. If this rendering technique is used, however, care must be taken to ensure that the T-vertices created by our algorithm do not cause shading anomalies. One way to handle this problem is to make a T-vertex a three-way vertex, with two collinear edges instead of a single edge orthogonal to the remaining edge. The element with the two collinear edges is then triangulated before shading is done.

7. 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

Figure 9: Rendered Boxes

Figure 10: Mesh for Boxes

only a few seconds. Times were measured using the UNIX *time* facility.

Figures 9 through 12 show two rendered images, along with their meshes. Timing data is given in Figure 13. All images were created with the small light source optimizations mentioned in Section 4.

8. 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 sufficient frequency.

Techniques presented here can easily be used to improve existing point-source renderers. Since the shading function for point sources is easy to compute, smooth, and differentiable, our mesh-refinement approach can easily be applied to improve image quality. Additionally, our area source techniques were designed to be simple and efficient enough that area illumination effects could be easily added to existing renderers.

There are several areas ripe for future work. A thorough examination of the space/time tradeoff involved in maintaining separate penumbra regions for each light source versus determining this on the fly when illuminating penumbras remains to be done. This also needs to be done to determine whether back-to-front ordering of the shadow testing and the concomitant light source splitting is actually more efficient than the basic set operation method for all scenes. The optimization techniques used by our algorithm could probably be improved by taking more advantage of the special properties of the problem, and better techniques for penumbra regions and global illumination functions need to be developed.

Figure 11: Rendered Chair

Figure 12: Mesh for Chair

Statistic	Figure	
Name	Boxes	Chair
Input Polygons	20	33
Output Polygons	1539	1360
Scene BSP Build (sec)	0.22	0.47
Source Splitting (sec)	0.20	0.20
Shadow Volume Build (sec)	9.32	8.47
Shadow Testing (sec)	0.68	2.98
Occluding Polygon Culling (sec)	0.07	0.23
Source Visibility Clipping (sec)	1129.68	971.65
Illumination Calculation (sec)	308.31	151.15
T Vertex Insertion (sec)	59.00	45.33
Total Time (sec)	1507.48	1150.48

Figure 13: Timing Statistics

9. ACKNOWLEDGEMENTS

We would like to thank John Howell and Alan Cline for their helpful discussions and suggestions. Thanks to the Applied Research Laboratory of the University of Texas at Austin for the use of their computing facilities. We would especially like to thank Walter S. Reed and Philip D. Heermann of ALFA Engineering, Inc. for encouragement and advice.

REFERENCES

[1] Amanatides, John, Ray Tracing with Cones, *Computer Graphics* (SIGGRAPH '84 Proceedings), Vol. 18, No. 3, July 1984, pp. 129–135.

[2] Baum, Daniel R., Holly E. Rushmeier and 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.

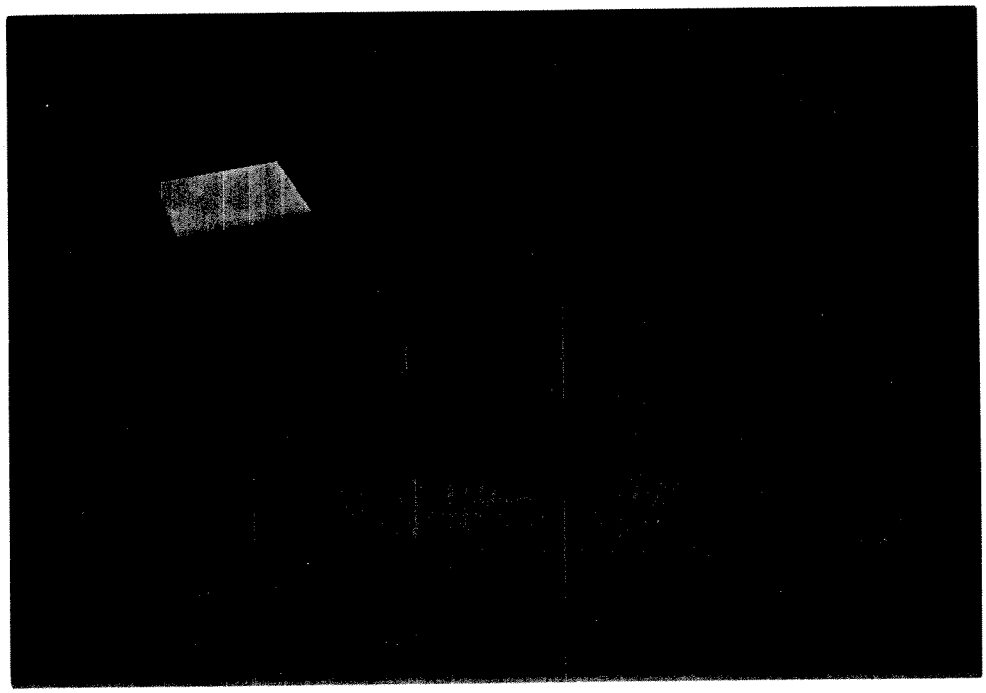

Figure 9

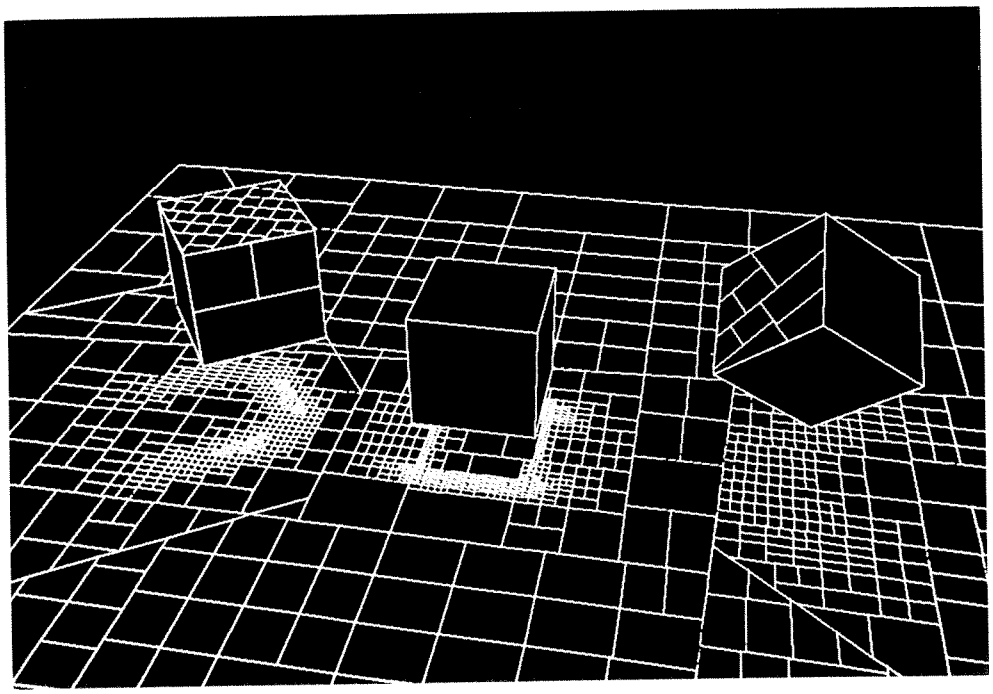

Figure 10

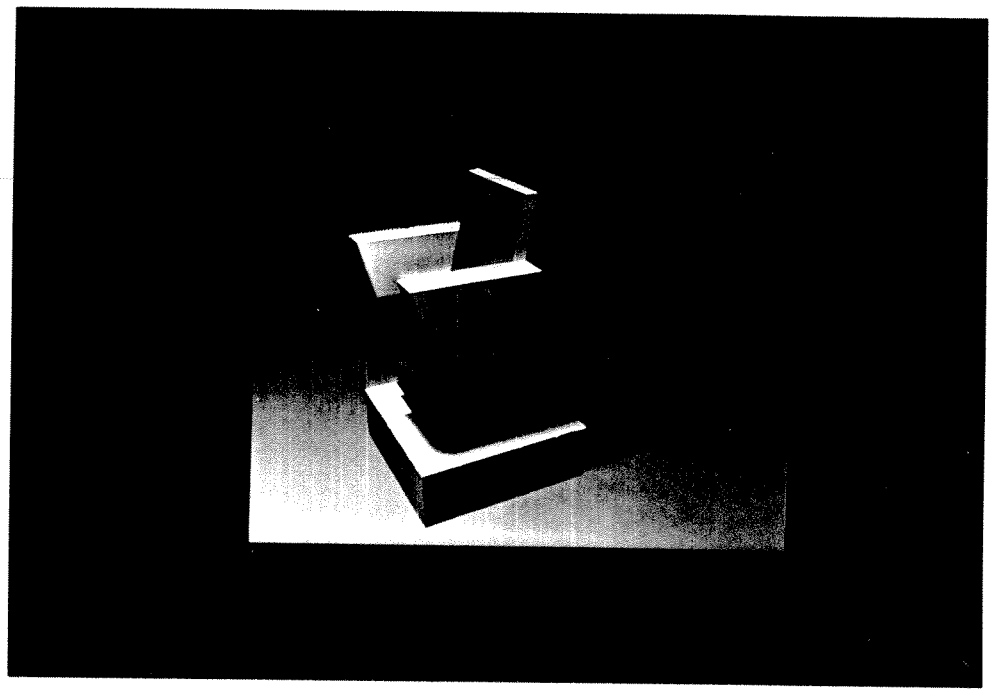

Figure 11

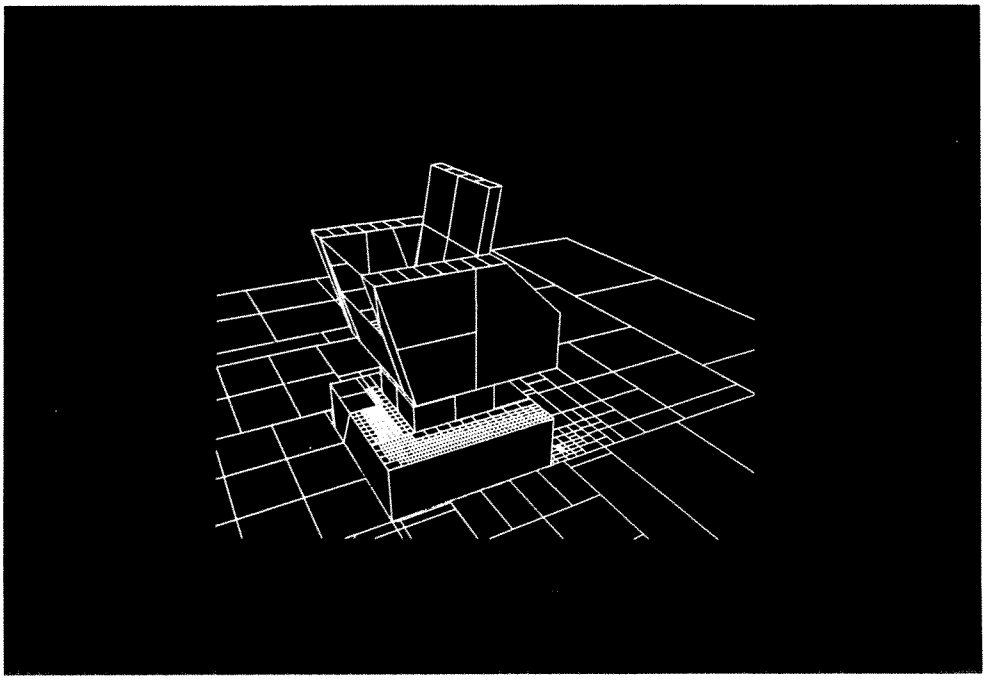

Figure 12

[3] Brotman, Lynne S. and Norman Badler, Generating Soft Shadows with a Depth Buffer Algorithm, *IEEE Computer Graphics and Applications* Vol. 4, No. 10, March 1984, pp. 71–81.

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

[5] Chin, Norman and Steven Feiner, Near Real-Time Shadow Generation Using BSP Trees, *Computer Graphics* (SIGGRAPH '89 Proceedings), Vol. 23, No. 3, July 1989, pp. 99–106.

[6] Cohen, Michael F. and 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.

[7] Cook, Robert L., Thomas Porter, and Loren Carpenter, Distributed Ray Tracing, *Computer Graphics* (SIGGRAPH '84 Proceedings), Vol. 18, No. 3, July 1984, pp. 137–145.

[8] Crow, Franklin, Shadow Algorithms for Computer Graphics, *Computer Graphics* (SIGGRAPH '77 Proceedings), Vol. 11, No. 3, August 1977, pp. 242–244.

[9] 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.

[10] Gouraud, Henri, Continuous Shading of Curved Surfaces, *IEEE Transactions on Computers*, C-20(6), June 1971, pp. 623–628.

[11] Haeberli, Paul, and Kurt Akeley, The Accumulation Buffer: Hardware Support for High-Quality Rendering, *Computer Graphics* (SIGGRAPH '90 Proceedings), Vol. 24, No. 4, August 1990, pp. 309–318.

[12] Haines, Eric A., and Donald P. Greenberg, The Light Buffer: A Shadow-Testing Accelerator, *IEEE Computer Graphics and Applications*, Vol. 6, No. 9, September 1986, pp. 6–16.

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

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

[15] Kajiya, James T., The Rendering Equation, *Computer Graphics* (SIGGRAPH '86 Proceedings), Vol. 20, No. 4, August 1986, pp. 143–150.

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

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

[18] Nishita, Tomoyuki and Eihachiro Nakamae, Half-Tone Representation of 3-D Objects Illuminated by Area Sources or Polyhedron Sources, *IEEE COMPSAC*, November 1983, pp. 237–242.

[19] Nishita, Tomoyuki and 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. 22–30.

[20] Press, William H., 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.

[21] Siegel, Robert and John R. Howell, *Thermal Radiation Heat Transfer*, Hemisphere Publishing Corp., Washington DC, 1981.

[22] Thibault, William and Bruce Naylor, Set Operations on Polyhedra Using Binary Space Partitioning Trees, *Computer Graphics* (SIGGRAPH 87 Proceedings), Vol. 21, No. 3, July 1987, pp. 153–162.

[23] Wallace, John R., Kells A. Elmquist, and Eric A. Haines, A Ray Tracing Algorithm for Progressive Radiosity, *Computer Graphics* (SIGGRAPH '89 Proceedings), Vol. 23, No. 3, July 1989, pp. 315–324.

[24] Woo, Andrew, Pierre Poulin, and Alain Fournier, A Survey of Shadow Algorithms, *IEEE Computer Graphics and Applications*, Vol. 10, No. 6, November 1990, pp. 13–32.