Fast Object-Precision Shadow Generation for Area Light Sources Using BSP Trees

Norman Chin
Steven Feiner

Department of Computer Science
Columbia University
New York, New York 10027

nc@cs.columbia.edu
feiner@cs.columbia.edu

Abstract

This paper introduces an efficient object-precision shadow generation algorithm for static polygonal environments directly illuminated by convex area light sources. Penumbra and umbra regions are calculated analytically and represented as a pair of BSP trees for each light source. As the trees are built, convex scene polygons are filtered down the trees, and split into fragments that are wholly lit, in penumbra, or in umbra. The illumination due to the light source is calculated at selected points within the wholly lit and penumbra regions by contour integration with the visible parts of the light source. We use a fast analytic algorithm to compute the fragments of the area light source visible from a point in penumbra. Rendering is done using hardware-supported linear interpolated shading on a 3D graphics workstation.

Because the scene itself is represented as a BSP tree, visible-surface determination may be performed by using either workstation-supported hardware (e.g., a z-buffer) or software BSP-tree traversal. We provide sample images created by our implementation, including timings and polygon counts.

CR Categories and Subject Descriptors: I.3.3 [**Computer Graphics**]: Picture/Image Generation—*Display algorithms*; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Constructive solid geometry (CSG)*; I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Color, shading, shadowing, and texture*

General Terms: Algorithms

Additional Keywords and Phrases: shadow volume, area light source, BSP tree, penumbra, umbra

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.
© 1992 ACM 0-89791-471-6/92/0003/0021...$1.50

Introduction

Shadow generation is a classic problem in 3D computer graphics that has been addressed by a wide variety of algorithms [13, 22]. Point light-source shadow algorithms essentially compute the visibility of parts of the environment from a point at the light source; therefore any point in the environment is either fully in or out of shadow. In contrast, in an environment lit by area light sources, a point in the environment may be either visible to the entirety of the light source, visible to no part of the light source (i.e., in the light source's *umbra*), or visible to only a portion of the light source (i.e., in the light source's *penumbra*). In this latter case, to compute the point's illumination, it is also necessary to determine which portions of the area light source are visible from the point. Since real light sources are not points and therefore cast both umbrae and penumbrae, an area light-source shadow algorithm can be used to create pictures that are more photorealistic in appearance than those created with a point light-source shadow algorithm.

Shadows from area light sources have been computed using radiosity approaches [9, 6], by summing the contributions of an approximating set of point light sources [5], by ray tracing shadow cones from points in a scene to spherical light sources [1], by distributed ray-tracing [10], and by an object-precision algorithm developed by Nishita and Nakamae [17]. With the exception of this single object-precision algorithm, all the other algorithms approximate the shadow boundaries on the objects in the scene. For each pair of a light source and a polyhedral object, Nishita and Nakamae compute the volume that the object fully shadows from the light source (its *umbra volume*) and the volume that the object partially shadows from the light source (its *penumbra volume*). The intersections of these volumes with the other objects in the environment are computed and guide the calculation of the illumination at selected points on the objects. For example, a point is fully shadowed if it is included in at least one umbra volume.

The algorithm that we describe here is inspired in part by this work; unlike Nishita and Nakamae, however, we build a single merged umbra volume and penumbra volume for each light source. Furthermore, these volumes are represented as

BSP trees [14, 15, 21, 16] using an efficient extension of the earlier BSP-tree-based shadow algorithm for point light sources [7]. Although subdivision is always done along exact shadow boundaries, further subdivision may be necessary to compute illumination more accurately. We have used both regular gridding and adaptive subdivision of fragments in the penumbra and wholly lit regions to compute the illumination at additional points.

Background

The binary space-partitioning (BSP) tree visible-surface algorithm was developed by Fuchs, Kedem, and Naylor [14], based in part on the work of Schumacker [19, 20]. A BSP tree defines a recursive partitioning of space by planes that embed the polygons in the scene. The tree's root is a polygon chosen from those in the scene. This polygon's plane partitions space into two half-spaces: the "positive" half-space contains all other polygons in front of the root's plane (on the side into which its normal points); the "negative" half-space contains all polygons behind the root's plane. If a polygon straddles the root's plane, it is cut by it and each of its pieces is assigned to the appropriate half-space. One polygon each from the positive and negative half-spaces are then selected to become the root's children. Each child is then recursively used to divide the remaining children in its half-space in the same way. The tree is complete when each leaf node contains a single polygon whose half-spaces are both empty. The BSP tree visible-surface algorithm is a modified inorder traversal of the scene's BSP tree, guided by a simple comparison of the eyepoint with each polygon's plane; this determines in $O(n)$ time a back-to-front ordering of the polygons for any eyepoint.

Thibault and Naylor [21] showed that BSP trees can be used to represent polyhedral solids. Each of the empty regions at the leaves is associated with a value of either "in" or "out". Assuming that each polygon that bounds a polyhedron has a normal that points out of the polyhedron, then an "in" region is bounded in part by the polygon's negative (back) half-space and an "out" region is bounded in part by the polygon's positive (front) half-space. The BSP tree's leaf nodes tessellate space into a set of convex polyhedral regions, a subset of which (the "in" regions) represent the solid.

The point light-source shadow algorithm described in [7, 8] uses BSP trees to model the polyhedral shadow volumes [11] cast by convex polygons. We call the BSP tree representation of the shadow volume the SVBSP (Shadow Volume BSP) tree. A regular BSP tree is first constructed for all polygons in the scene. (Note that if the scene is modified, then the scene BSP tree must be recalculated.) The scene BSP tree allows the shadow algorithm to obtain all scene polygons efficiently in front-to-back order relative to an arbitrary point light source. Only scene polygons that face the light are selected. The point light source and the first scene polygon chosen define together a shadow volume that is a semi-infinite pyramid. Each of the pyramid's faces is embedded in a plane defined by the light source and an edge of the scene polygon. A point will be in shadow if it lies within the pyramid and in the scene polygon's negative half-space. The scene polygon is itself fully lit.

Because of the front-to-back order imposed by the BSP tree traversal, each new scene polygon processed is guaranteed not to block any of the previously selected scene polygons from the light. It may be wholly or partially in shadow itself, however. To determine which parts of the new polygon are visible from the light source, we must partition the polygon into parts that are inside and outside the current SVBSP-tree shadow volume. Note that there is no need to compare the new polygon with the planes that embed the previous scene polygons, since the BSP-tree traversal order ensures that the new polygon does not lie between the light source and the preceding scene polygons. Those parts of the new polygon that are inside the shadow volume are in shadow; those parts that are outside it are lit. Furthermore, any parts that are outside define additional shadow volumes that must be added to the SVBSP tree. The point light-source algorithm efficiently combines these two steps of classifying polygon fragments and enlarging the SVBSP tree by using a simplified version of the Boolean set union operation algorithm presented in [21]. Each remaining polygon is processed in this fashion to determine which of its parts are shadowed.

Like the BSP-tree point light-source shadow algorithm, our BSP-tree convex area light-source algorithm supports multiple light sources. The area light-source algorithm extends the point light-source algorithm by classifying polygons into fragments that are wholly lit, in penumbra (partially blocked from the light source), or in umbra (wholly blocked from the light source). To do this, we must first define the umbra and penumbra volumes of an area light source.

Constructing Penumbra and Umbra Volumes

In environments composed of convex polygons illuminated by convex light sources, the penumbra and umbra volumes associated with a single scene polygon can be constructed entirely from three kinds of planes:

- *scene polygon planes*, a single one of which is defined by the scene polygon itself.
- *light-source vertex planes*, defined by a vertex of the light source and an edge of a scene polygon, oriented so that the scene polygon is entirely in the plane's negative half-space or on the plane.
- *light-source edge planes*, defined by an edge of the light source and a vertex of a scene polygon, oriented so that the scene polygon is entirely in the plane's negative half-space or on the plane.

We use Nishita and Nakamae's criteria for determining those planes that define the penumbra and umbra volumes of a scene polygon. The penumbra volume is the intersection of the scene polygon's negative half-space with the negative half-spaces of certain light-source vertex planes and light-source edge planes. These light-source vertex planes and light-source edge planes are those for which the vertices of the light source are entirely in the plane's positive half-space or on the plane. (The penumbra volume actually encloses points in umbra, as well as those in penumbra.)

Figure 1 shows the penumbra cast on a large polygon by a triangle light source illuminating a quadrilateral scene polygon. Dashed lines passing from each light source vertex to

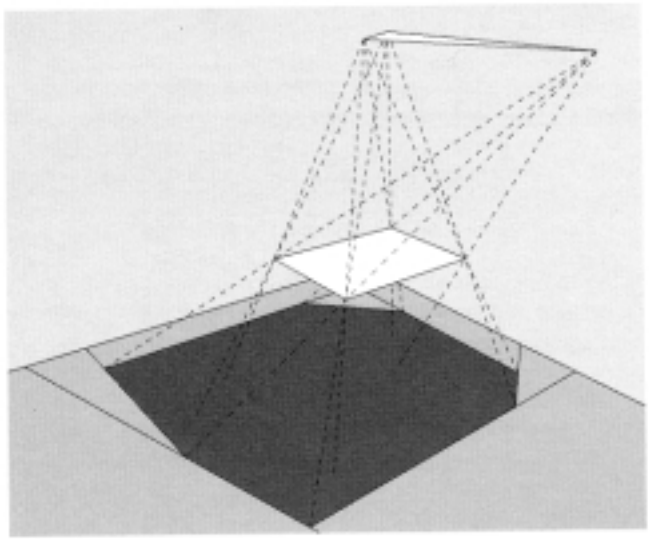


Figure 1: Penumbra of area light source, with light-source vertex planes and light-source edge planes.

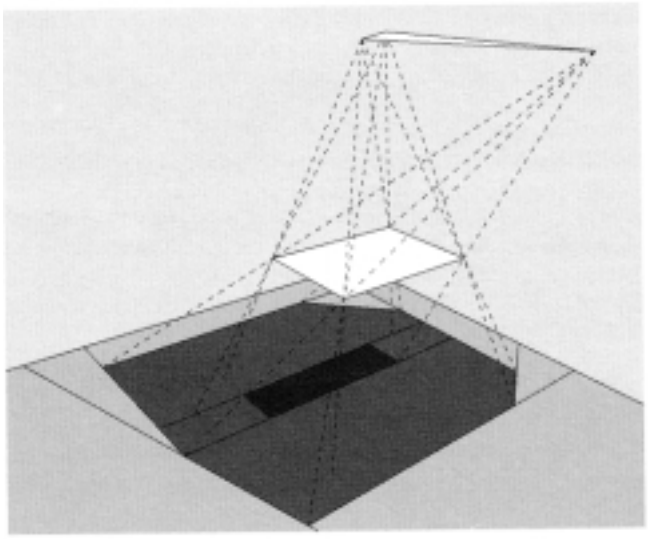


Figure 2: Penumbra and umbra, with light-source vertex planes and light-source edge planes.

all scene polygon vertices define the light-source vertex planes and light-source edge planes. (The additional fragmentation surrounding the penumbra outline is caused by the algorithm's classification process, which we describe later.) Note that the planes that bound the penumbra volume are those that have the light source in their positive half-space and the scene polygon in their negative half-space. Thus, any point in the positive half-space of such a plane cannot be blocked from any part of the light source by the scene polygon.

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

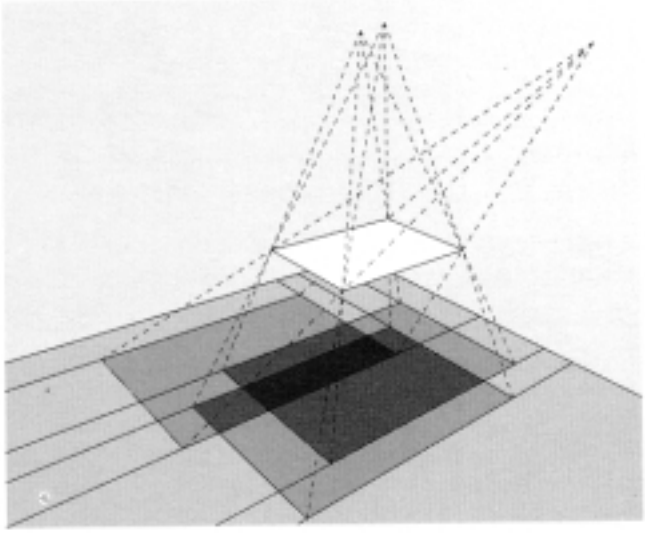


Figure 3: Shadows cast by 3 point light sources at the vertices of an area light source.

Figure 2 shows the same scene as Figure 1 with the umbra included. Note that the dashed lines that lie in the planes that define the umbra outline do not always pass through the umbra outline's vertices.

Figure 3 shows an alternative, but exactly equivalent, way to define the umbra and penumbra volumes. They can be derived from the shadow volumes generated when the convex scene polygon is illuminated by point light sources at the convex area light source's vertices. (The additional fragmentation of the ground plane is caused by the BSP-tree point light-source shadow algorithm used to create this figure.) The area light source's umbra volume contains those points that are blocked from all of the area light source's vertices. This corresponds to the intersection of the point light-source shadow volumes, which is defined by the set of light-source vertex planes specified previously.

The union of the point light-source shadow volumes encloses all points that are blocked from one or more vertices of the area light source. This is only a subset of the light source's penumbra volume, however, since it does not include those points that are visible from all the area light source's vertices, but are blocked from part of the area light source's interior. It can be shown that to enclose these points the penumbra volume must be the convex hull of the point light-source shadow volumes. The convex hull is defined by the set of light-source vertex planes and light-source edge planes specified previously.

Overview

Instead of the single SVBSP tree required by the point light-source shadow algorithm, we use two BSP trees: a *penumbra tree* and an *umbra tree* [8]. Each BSP-tree internal node is defined by a light-source vertex plane or light-source edge plane.

Much like the point light-source shadow algorithm, two steps must be performed for each scene polygon:

- Classifying the polygon into wholly lit, penumbra, and umbra fragments.
- Enlarging the penumbra and umbra trees with light-source vertex planes and light-source edge planes defined by the polygon.

The classified fragments must then be illuminated and scan-converted.

Algorithm

Preprocess. An obvious approach to classification would be to compare each scene polygon with the shadow volume of every other scene polygon. However, polygons that are not in the same half-space of a polygon as the light source cannot cast shadows on that polygon or any other polygon in the light source's half-space. Therefore, as in the point light-source shadow algorithm, we first compute a BSP tree for the entire scene. This allows us to perform a modified inorder traversal of the tree to process scene polygons in front-to-back order relative to the light source.

Unlike a point light source, an area light source may not lie entirely in a single half-space of a scene polygon. If this occurs, choosing different points on the area light source will generate different BSP-tree traversal orders. To obtain a unique order, we first split each area light source by those scene polygons that intersect it and that are in the lit half-space of the light source's plane. Since each of the resulting light sources is wholly on one side of each scene polygon, any point within the light source will generate the same front-to-back ordering of the scene polygons. For convenience, we pick the centroid of each resulting area light source as the point from which to compute the ordering. We must also ensure that each scene polygon that straddles a light source plane is split by the plane.

Classification. Classification and tree enlargement are interleaved as they are performed incrementally for each scene polygon in front-to-back order. Therefore, the two shadow trees represent the merged penumbra and umbra volumes of all the scene polygons processed thus far. Classification occurs by filtering each polygon down one or both shadow trees. This process is applied recursively until all of a polygon's fragments reach the "in" and "out" leaves.

A polygon is first filtered down the penumbra tree. Any fragment that reaches an "out" cell is marked as wholly lit and will not be compared with the umbra tree. (Recall that the umbra volume is wholly contained within the penumbra volume, so any fragment outside the penumbra volume cannot be in umbra.) Any fragment that reaches an "in" cell is at least in penumbra and may be in umbra. Each such fragment must then be filtered down the umbra tree. Any fragment that reaches an umbra tree "out" cell is in penumbra, whereas any fragment that reaches an umbra tree "in" cell is in umbra. The penumbra and umbra BSP trees are enlarged by unioning them with the penumbra volume and umbra volume, respectively, defined by the full scene polygon. We trivially classify as in umbra any polygon that is in the back half-space of a light source, without any need for filtering. In addition, if we assume that polygons are "one-sided" and that they bound closed polyhedra, we can also trivially classify as in umbra all polygons that are back-facing relative to the light source.

As in the earlier point light-source algorithm, multiple area light sources are supported by pipelining. The fragments classified relative to one light source must be used as input to the algorithm when processing the next light source. Thus, when all light sources have been processed, each of the output fragments is uniquely classified relative to each of the light sources. (See the pseudocode for the algorithm in the appendix.)

Example. Figure 4 shows how the algorithm handles a simple example. For ease of explanation, the figure is drawn in 2D and thus shows umbra and penumbra areas cast by a linear light source on lines in the plane. (In 2D, only light-source vertex edges are needed, but the definitions are the same otherwise.)

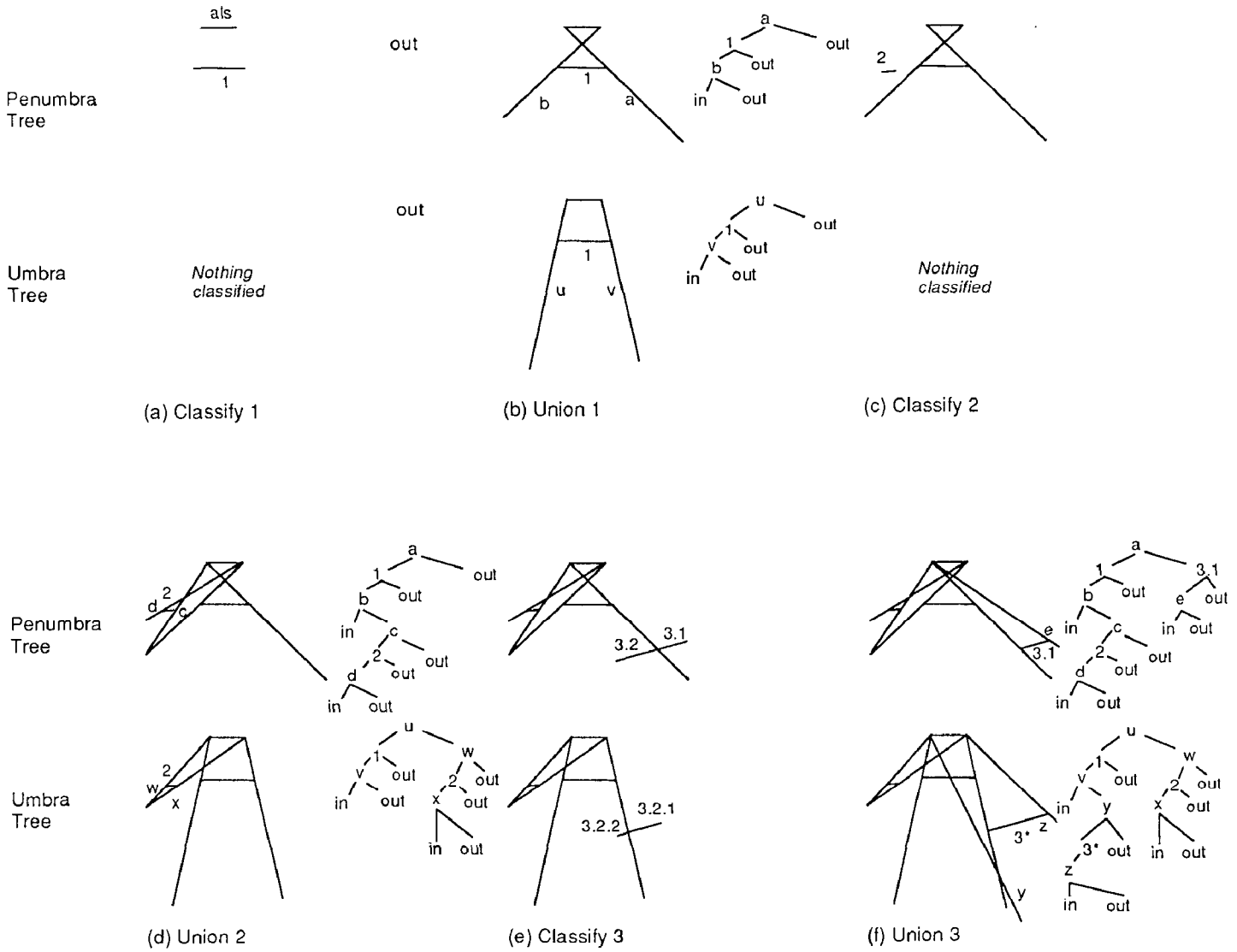
Initially, both shadow trees are null ("out"), as shown in Figure 4(a). Polygon 1 is first filtered down the penumbra tree and is trivially classified as fully lit. Because no part of the polygon was classified as in penumbra, no classification is done using the umbra tree. Next, as shown in Figure 4(b), polygon 1's penumbra is used to enlarge the penumbra tree. Rather than using the many lit fragments that may have been identified, the original polygon is used instead. In 2D, this results in a union with polygon 1 and light-source vertex planes *a* and *b*, which define polygon 1's penumbra volume. Although polygon 1 was not classified using the umbra tree, it must be used to enlarge the umbra tree and results in a union with volume defined by polygon 1 and the light-source planes *u* and *v*.

Next, polygon 2 is classified, as shown in Figure 4(c). Much like polygon 1, polygon 2 is classified as wholly lit relative to the penumbra tree and is not classified using the umbra tree. The penumbra tree is then enlarged with polygon 2 and planes *c* and *d*, and the umbra tree is enlarged using polygon 2 and planes *w* and *x*. (Figure 4d). Unlike polygon 1, however, polygon 2's addition to the merged umbra volume is not semi-infinite.

Polygon 3 is more interesting. When it is classified against the penumbra tree, as shown in Figure 4(e), it is split by face *a* into fragments 3.1 and 3.2. Fragment 3.1 is classified as "out" (i.e., wholly lit), while fragment 3.2 is classified as "in" (i.e., in some combination of penumbra and umbra). Therefore, only fragment 3.2 must be filtered down the umbra tree. When this is accomplished, the umbra tree's *v* plane further subdivides fragment 3.2 into fragments 3.2.1 (in penumbra) and 3.2.2 (in umbra). At this point, both shadow trees are enlarged using the original polygon 3, as shown in Figure 4(f). This results (in 2D) in the polygon fragment 3.1 and plane *e* being added to the penumbra BSP tree and a volume defined by planes *y* and *z* and 3*, the fraction of polygon 3 not in umbra, being added to the umbra BSP tree.

Illumination

After classifying all fragments by all light sources, we need to illuminate them. We use an analytic direct diffuse illumination model [17] based on contour integration, which is evaluated at polygon vertices within the penumbra and wholly lit regions. Unlike full global illumination algorithms, interreflections are not computed. Points in umbra are lit by an ambient light component alone. In our implementation, interpolated shading is performed using 3D graphics hardware.



(d) Union 2 (e) Classify 3 (f) Union 3

Figure 4: Classifying polygons and enlarging the penumbra and umbra BSP trees. Parts (a–f) show penumbra and umbra volumes (areas) and their trees during the classification of three polygons (lines).

Although the classification process divides polygons along precise shadow boundaries, large polygons may remain that are homogeneously lit or in penumbra. While direct illumination should vary continuously across these surfaces, linear interpolation does not adequately represent these changes and does not allow any polygon interior pixel to be brighter than the polygon's vertices. Therefore, illumination must be computed at additional points within the scene. In the pictures included here, we subdivide wholly lit and penumbra regions using regular grids of user-specified granularity. We generally use a finer grid in the penumbra region, since the intensity typically changes more quickly than in an equivalent wholly lit region. The umbra region is not subdivided because it receives only constant ambient illumination. Subdivision is performed after classification, since it has no effect on the precision at which classification occurs and would increase the classification overhead if performed first. BSP-tree subdivision can often generate thin sliver polygons that can cause shading anomalies. Better results would be obtained with an adaptive subdivision algorithm that attempted to generate well-shaped fragments from these potentially problematic fragments [3].

Diffuse illumination equation. To determine the illumination at a point that is wholly lit, we perform contour integration with the light source from the point being lit, as described in [17]. The diffuse illumination at point p due to the light-source is computed as

$$I_p = \frac{I_l}{2} \sum_{v=1}^{n} \alpha_v \cos(\beta_v),$$

where I_l is the light source intensity, n is the number of vertices of the light source, α_v is the angle between the vector from p to light-source vertex v and the vector from p to light-source vertex v+1, and β_v is the angle between the plane defined by the two vectors used to compute α and the plane on which p lies. (The cosine of β_v may be computed as the dot product of the normalized surface normal at p with the cross product of the normalized vectors used to define α_v.)

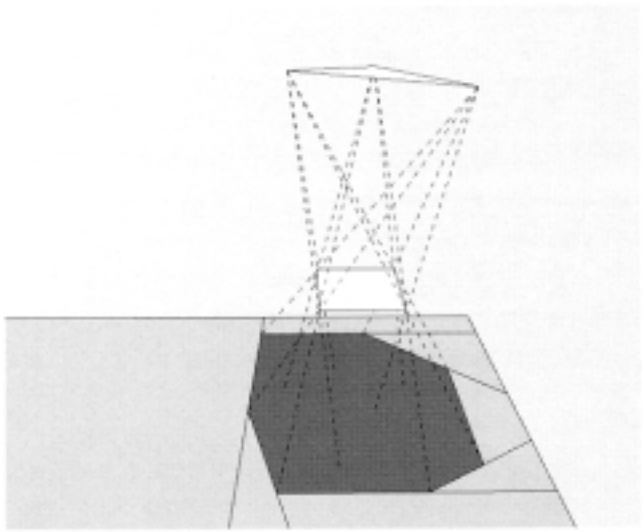


Figure 5: Penumbra volume of a single polygon.

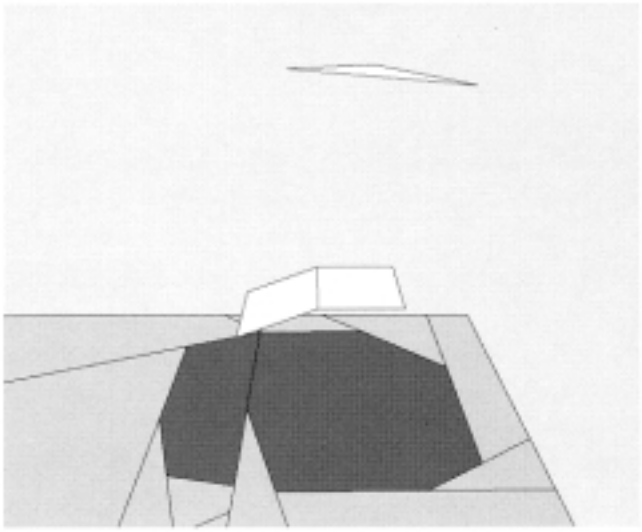


Figure 6: Incorrect merged penumbra volume of two polygons.

Analytic visibility for penumbra vertices. For points in penumbra, we must determine the fragments of the light source that are visible from the point. We accomplish this with a simplified version of the earlier point light-source shadow algorithm. By traversing the scene BSP tree, we can obtain all polygons between the point whose illumination is being computed and the plane of the light source. (Whether the traversal order is back-to-front or front-to-back is unimportant.) As before, we consider only those scene polygons that are front-facing relative to the light source (i.e., back-facing relative to the point being illuminated).

For each scene polygon, we clip the light-source polygon by the point light-source shadow volume defined by the point in penumbra and the edges of the scene polygon. The portion of the light source that is inside this volume is discarded and the portions that are outside are retained for comparison with the next scene polygon's volume. (Since the original light-source polygon bounds any light-source fragments produced, it can be used to do an extent check if desired.) The fragments remaining when the BSP-tree traversal encounters the light-source polygon are those that are visible from the point in penumbra and we sum the illumination contributed by each light-source fragment.

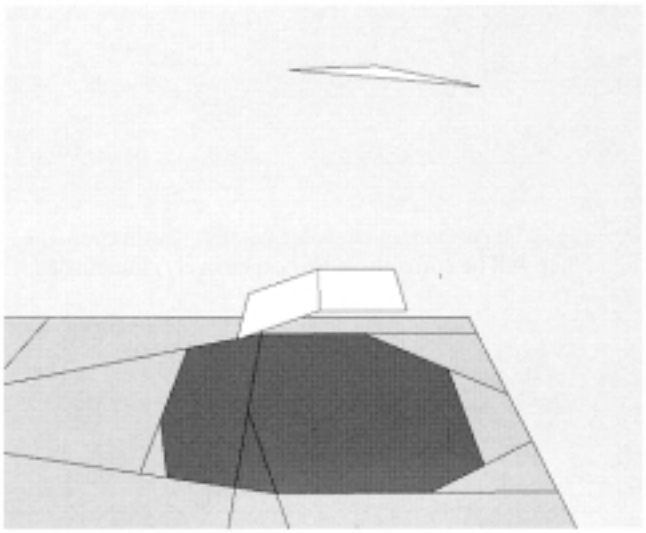


Figure 7: Correct merged penumbra volume of two polygons.

Discussion and Implementation

In the BSP-tree point light-source algorithm, the SVBSP tree was enlarged to reflect a polygon's contribution to the shadow volume by using a simplified version of the set union algorithm described in [21]. This simplification ignored any part of a polygon that fell within the existing volume. It used only planes determined by those fragments of the polygon that were wholly lit. For a point light source, the volume determined by these planes is guaranteed not to intersect the existing shadow volume. (In other words, no fragment lit by a point light source casts a shadow that falls within the shadow cast by any other lit fragment.) This is not the case for penumbra volumes, however. The penumbra volume cast by one polygon may intersect the volume cast by another. Therefore, a regular BSP-tree set union operation [21] must be performed.

Figure 5 shows the penumbra volume defined by a single scene polygon. Figure 6 shows the incorrect results that occur if a second scene polygon is added and the planes defining its penumbra volume are not continued into the penumbra volume of the original polygon. In this case, the penumbra volume of the second polygon considered by itself is similar to that of the first polygon and overlaps the first polygon's penumbra volume. This new penumbra volume crosses over the leftmost light-source vertex plane bounding the first polygon's penumbra volume. Part of the second polygon's contribution to the merged penumbra volume is ignored, resulting in the penumbra gap shown at the bottom of the figure. Figure 7 shows the correct merged penumbra volume that results when the original volume is enlarged properly by unioning the second polygon's penumbra volume with the current penumbra BSP tree, taking into account the possibility of

Plate	# lights	input polygons	shadow time (sec)	grid time (sec)	illum time (sec)	total time (sec)	output polygons	actual vertices	illuminated vertices
1,2	1	14	0.3	0.6	2.2	3.1	747	1341, 1668, 231	494, 437, 61
3,4	1	149	13.4	5.3	105.4	124.1	3348	3125, 8556, 3043	1576, 2512, 839
5	2	70	8.6	10.2	30.8, 37.9	87.5	4085	6768, 4627, 5791	2211, 1140, 1823
								7969, 5847, 3370	2608, 1429, 1132
6	2	151	35.8	23.2	178.1, 247.2	484.3	9344	10821, 14938, 13458	3904, 3839, 4008
								12599, 16594, 10024	4352, 4424, 2944

Figure 8: Statistics for color plates. All timings are given in elapsed wall-clock seconds for an HP 9000 380 (22 MIPS, 2.6 MFLOPS). Input polygon count takes into account splits caused by building the scene BSP tree. Shadow time is the time to classify the input polygons. Grid time is the time to subdivide the wholly lit and penumbra regions to produce the output polygons. Illumination time is the time to determine illumination values for the output vertices. Actual vertices lists the numbers of wholly lit, penumbra, and umbra vertices. Illuminated vertices lists the numbers of wholly lit, penumbra, and umbra calculations performed, which is lower than the actual vertex count because of vertex sharing. (Figures 5 and 6 have one illumination time for each light source, and one set of vertex statistics for each light source. Note that the sum of the actual wholly lit, penumbra, and umbra vertices is the same for each light source in these figures.)

overlapping volumes.

If the penumbra volume were incomplete, fragments that were contained in the volume's missing parts would be marked as wholly lit and would be incorrectly illuminated. Therefore, it is essential that the entirety of the actual penumbra volume be represented. In contrast, since the umbra volume is contained within the penumbra volume, if the umbra volume were incomplete, fragments that were contained in the umbra volume's missing parts would be marked as being in penumbra. Since the illumination algorithm correctly determines that these fragments are wholly blocked from the light, they will be correctly (albeit expensively) illuminated.

It is interesting to note that unioning each polygon's umbra volume with the existing umbra volume does *not* create the complete set of all points that are fully blocked from the light source. Instead, it creates the set of all points p such that there is at least one polygon that fully blocks p from the light source. That is, the union of the individual polygon umbra volumes does not contain those points that are fully blocked from the light source only because of the contributions of multiple blocking polygons. An example of this occurs in Figure 4(f). Points in the gap between planes v and y at the bottom of the volume are fully blocked from the light source because of the combined effect of polygons 1 and 3, yet do not lie in the merged umbra volume.

As with most analytic algorithms, care must be taken to contend with finite floating-point precision. To avoid problems, as polygons are split, the plane equations are copied, not recomputed. A similar method can be used to guarantee that split edges remain truly collinear. When a polygon edge is split, we also insert the new vertex in any other polygon that shares the edge. This prevents the shading discontinuities that would be caused by a "T" vertex. The vertex at which a split occurs is also shared among the polygon's fragments. This allows each vertex's illumination computation to be performed only once. It also makes it easy to determine the kinds of fragments that share a given vertex. If a vertex is shared by a wholly lit fragment and a penumbra fragment, we treat the vertex as wholly lit for both, eliminating the need for the light-source visibility test. If a vertex is shared by a wholly lit fragment and an umbra fragment, it is treated differently in each to preserve the boundary. We currently do not promote vertices shared by both penumbra and umbra fragments to umbra vertices. This avoids the possibility of smearing a full umbra shadow into a penumbra fragment when the umbra fragment is blocked by an object that does not block the penumbra fragment. This is similar to the problem of "light leaks" [6], in which a polygon is straddled by a partition that blocks light from some of its vertices, even though illumination leaks under the partition through interpolated shading.

Another possible optimization that would reduce fragmentation is to merge fragments together when both subtrees were classified as "in" or or as "out" [8]. Since a penumbra volume extends infinitely far past the object that casts it, we have also considered some approaches to restricting its extent, similar to Bergeron's use of end caps on shadow volumes to eliminate the need to perform shadow computations outside of a light's "sphere of influence." [4].

The area light-source algorithm has been implemented in C on an HP 9000 380 TurboSRX workstation, and the results are displayed interactively using hardware interpolated shading. Because the scene polygons are represented as a BSP tree, either the hardware z-buffer or a software BSP-tree visible-surface algorithm can be used to render the scene.

Pictures. Color Plate 1 shows two objects floating in air and one triangle light source with their penumbra and umbra regions. The light grey and dark grey fragments are in penumbra and umbra respectively, while the colored fragments are wholly lit. The wholly lit and penumbra fragments have been gridded after classification. Note the band of penumbra separating the umbra regions of both objects. As described above, this strip should be in umbra, but will be properly illuminated because the illumination computation determines that its vertices are unlit. The same scene after illumination and interpolated shading is shown in Color Plate 2.

Color Plate 3 shows a room with one quadrilateral area light source and gray fragments to represent the regions identified as being in penumbra and umbra. Color Plate 4 shows the room as it appears after illumination and shading. Color Plate 5 shows a different view of a simpler version of the room without the playpen, illuminated by two quadrilateral light sources. Color Plate 6 shows the same scene as Color

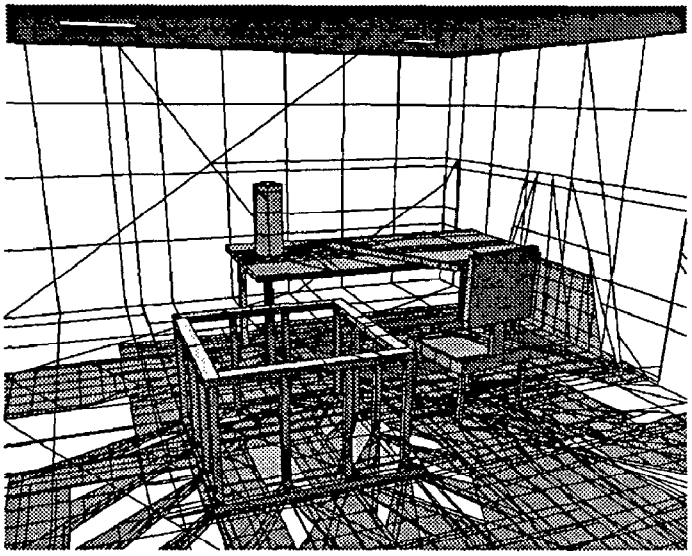


Figure 9: Room scene classified, 2 lights.

Plate 4, illuminated by both light sources. Figure 8 provides statistics for the color plates. Figure 9 shows the room depicted in Color Plate 6, prior to illumination, with the fragments produced by classification with both light sources and gridding.

Note that the most expensive part of the algorithm is the illumination phase, which need not be accomplished if the user is interested only in classifying objects according to their visibility, which is necessary in a number of applications in areas such as computer vision and graphics [12].

Conclusions and Future Work

The algorithm described here analytically generates penumbra and a subset of the umbra for static convex polygonal environments illuminated by convex area light sources. It is relatively simple to implement, places no restrictions on the location of objects and light sources, and runs efficiently for small scenes on modern workstations with hardware 3D graphics support. To generate further points at which illumination is sampled, we have implemented both regular gridding and simple adaptive subdivision of those fragments that are wholly lit or in penumbra.

We believe that an efficient analytic shadow algorithm would be useful in multiple passes of a radiosity approach (not just for the initial light-source calculations, as implemented in [18]). If selected radiators were treated as area light sources, object-precision shadow boundaries could be determined, instead of the relatively coarse boundaries obtained with current adaptive meshing techniques. This may make it possible to create more accurate images, with the illumination contour integral used to calculate analytic form factors [2] that properly take into account obstructions, guided by the shadow (i.e., visibility) classification phase.

Acknowledgments

This work was supported in part by the Office of Naval Research under Contract N00014-91-J-1872, the Defense Advanced Research Projects Agency under Contract N00039-84-C-0165, and an equipment grant from the Hewlett-Packard Company. Thanks to Clark Still and Tim Lee of the Columbia University Department of Chemistry, and Marilyn Noz of the NYU School of Medicine for generously allowing us to use their workstations.

References

1. Amanatides, J. Ray Tracing with Cones. *Proc. SIGGRAPH '84* (Minneapolis, MN, July 23–27, 1984). In *Computer Graphics*, 18(3), July 1984, 129–135.

2. Baum, D., Rushmeier, H., and Winget, J. Improving Radiosity Solutions Through the Use of Analytically Determined Form-Factors. *Proc. SIGGRAPH '89* (Boston, MA, July 31–August 4, 1989). In *Computer Graphics*, 23(3), July 1989, 325–334.

3. Baum, D., Mann, S., Smith, K., and Winget, J. Making Radiosity Usable: Automatic Preprocessing and Meshing Techniques for the Generation of Accurate Radiosity Solutions. *Proc. SIGGRAPH '91* (Las Vegas, NV, July 28–August 2, 1991). In *Computer Graphics*, 25(4), July 1991, 51–60.

4. Bergeron, P. A General Version of Crow's Shadow Volumes. *IEEE CG&A*, 6(9), September 1986, 17–28.

5. Brotman, L. and Badler, N. Generating Soft Shadows with a Depth Buffer Algorithm. *IEEE CG&A*, 4(10), October 1984, 5–12.

6. Campbell, A.T., III, and Fussell, D.S. Adaptive Mesh Generation for Global Diffuse Illumination. *Proc. SIGGRAPH '90* (Dallas, TX, August 6–10, 1991). In *Computer Graphics*, 24(4), August 1990, 155–164.

7. Chin, N. and Feiner, S. Near Real-Time Shadow Generation Using BSP Trees. *Proc. SIGGRAPH '89* (Boston, MA, July 31–August 4, 1989). In *Computer Graphics*, 23(3), July 1989, 99–106.

8. Chin, N. *Near Real-Time Object-Precision Shadow Generation Using BSP Trees.* MS Thesis, Dept. of Computer Science, Columbia University, New York, NY, 1990.

9. Cohen, M.F. and Greenberg, D.P. The Hemi-Cube: A Radiosity Solution for Complex Environments. *Proc. SIGGRAPH '85* (San Francisco, CA, July 22–26, 1985). In *Computer Graphics*, 19(3), July 1985, 31–40.

10. Cook, R.L., Porter, T., and Carpenter, L. Distributed Ray Tracing. *Proc. SIGGRAPH '84* (Minneapolis, MN, July 23–27, 1984). In *Computer Graphics*,

18(3), July 1984, 137–145.

11. Crow, F. Shadow Algorithms for Computer Graphics. *Proc. SIGGRAPH '77* (San Jose, CA, July 20–22, 1977). In *Computer Graphics*, 11(2), Summer 1977, 242–248.

12. Feiner, S. and Seligmann, D. Dynamic 3D illustrations with visibility constraints. In Patrikalakis, N. (ed.), *Scientific Visualization of Physical Phenomena (Proc. Computer Graphics International '91*, Cambridge, MA, June 26–28, 1991), Springer-Verlag, Tokyo, 1991, 525–543.

13. Foley J., van Dam, A., Feiner, S., and Hughes, J. *Computer Graphics: Principles and Practice, Second Edition*, Addison-Wesley, Reading MA, 1990.

14. Fuchs, H., Kedem, A., and Naylor, B. On Visible Surface Generation by A Priori Tree Structures. *Proc. SIGGRAPH '80* (Seattle, WA, July 14–18, 1980). In *Computer Graphics*, 14(3), July 1980, 124–133.

15. Fuchs, H., Abram, G., and Grant, E. Near Real-Time Shaded Display of Rigid Objects. *Proc. SIGGRAPH '83* (Detroit, MI, July 25–29, 1983). In *Computer Graphics*, 17(3), July 1983, 65–72.

16. Naylor, B., Amanatides, J., and Thibault, W. Merging BSP Trees Yields Polyhedral Set Operations. *Proc. SIGGRAPH '90* (Dallas, TX, August 6–10, 1991). In *Computer Graphics*, 24(4), August 1990, 115–124.

17. Nishita, T. and Nakamae, E. Half-Tone Representation of 3-D Objects Illuminated by Area Sources or Polyhedron Sources. *Proc IEEE COMPSAC*, November 1983, 237–241.

18. Nishita, T. and Nakamae, E. Continuous Tone Representation of Three-Dimensional Objects Taking Account of Shadows and Interreflection. *Proc. SIGGRAPH '85* (San Francisco, CA, July 22–26, 1985). In *Computer Graphics*, 19(3), July 1985, 23–30.

19. Schumacker, R., Brand, B., Gilliland, M., and Sharp, W. Study for Applying Computer-Generated Images to Visual Simulation. Technical Report AFHRL-TR-69-14, NTIS AD700375, US Air Force Human Resources Lab, Air Force Systems Command, Brooks AFB, TX, September 1969.

20. Sutherland, I., Sproull, R., and Schumacker, R. A Characterization of Ten Hidden-Surface Algorithms. ACM Computing Surveys, 6(1), March 1974, 1–55.

21. Thibault, W. and Naylor, B. Set Operations on Polyhedra Using Binary Space Partitioning Trees. *Proc. SIGGRAPH '87* (Anaheim, CA, July 27–31, 1987). In *Computer Graphics*, 21(4), July 1987, 153–162.

22. Woo, A., Poulin, P. and Fournier, A. A Survey of Shadow Algorithms. *IEEE CG&A*, 10(6), November 1990, 13–32.

Appendix: Pseudocode

```
procedure generateShadows (ALSlist, BSPtree)

  for each node n in BSPtree ; scene BSP tree
    copy n.scenePolygon into n.fragmentList
  endfor

  for each als in ALSlist
    centroid := centroid of als

    pBSP := OUT_CELL ; penumbra BSP tree
    uBSP := OUT_CELL ; umbra BSP tree

    for each node n in BSPtree in front-to-back order
        relative to centroid

      ; move n.fragmentList to fragmentList
      ; so that n.fragmentList can be recreated
      ; with fully classified and subdivided fragments
      fragmentList := n.fragmentList
      n.fragmentList := NULL

      for each fragment f in fragmentList
        if f not facing centroid OR als not facing f
          mark f in umbra
          n.fragmentList := append(n.fragmentList,f)
        else
          ; split f into wholly lit & shadowed fragments
          ; by filtering down pBSP

          tempFragmentList := NULL
          classifyWhollyLitOrShadowed
            (als,pBSP,f,&tempFragmentList)

          ; partition shadowed fragments into penumbra
          ; and umbra

          for each fragment t in tempFragmentList
            if t is shadowed
              classifyPenumbraOrUmbra
                (als,uBSP,t,&n.fragmentList)
            else
              n.fragmentList :=
                append(n.fragmentList,t)
            endif
          endfor

          ; enlarge pBSP and uBSP trees

          pv :=
            constructPolygonPenumbra(als,n.scenePolygon)
          pBSP := union(pBSP,pv) ; see [21]
          uv :=
            constructPolygonUmbra(als,n.scenePolygon)
          uBSP := union(uBSP,uv)
        endif
      endfor ; fragment
    endfor ; node

    discard pBSP and uBSP

  endfor ; als
endproc
```

```
procedure classifyWhollyLitOrShadowed
    (als,pBSP,f,fragmentList)

  if (pBSP is a leaf)
    if (pBSP == OUT_CELL)
      mark f as wholly lit
    else
      mark f as shadowed
    endif
    fragmentList := append(fragmentList,f)
  else
    splitPolygon(pBSP.plane,f,&negPart,&posPart)
    if (negPart != NULL)
      classifyWhollyLitOrShadowed(als,pBSP.negChild,
        negPart,&fragmentList)
    endif
    if (posPart != NULL)
      classifyWhollyLitOrShadowed(als,pBSP.posChild,
        posPart,&fragmentList)
    endif
  endif

endproc
```

```
procedure classifyPenumbraOrUmbra
    (als,uBSP,f,fragmentList)

  if (uBSP is a leaf)
    if (uBSP == OUT_CELL)
      mark f as penumbra
    else
      mark f as umbra
    endif
    fragmentList := append(fragmentList,f)
  else
    splitPolygon(uBSP.plane,f,&negPart,&posPart)
    if (negPart != NULL)
      classifyPenumbraOrUmbra(als,uBSP.negChild,
        negPart,&fragmentList)
    endif
    if (posPart != NULL)
      classifyPenumbraOrUmbra(als,uBSP.posChild,
        posPart,&fragmentList)
    endif
  endif

endproc
```