Near Real-Time Shadow Generation Using BSP Trees

Norman Chin
Steven Feiner

Department of Computer Science
Columbia University
New York, NY 10027

Abstract

This paper describes an object-space shadow generation algorithm for static polygonal environments illuminated by movable point light sources. The algorithm can be easily implemented on any graphics system that provides fast polygon scan-conversion and achieves near real-time performance for environments of modest size. It combines elements of two kinds of current shadow generation algorithms: two-pass object-space approaches and shadow volume approaches. For each light source a Binary Space Partitioning (BSP) tree is constructed that represents the shadow volume of the polygons facing it. As each polygon's contribution to a light source's shadow volume is determined, the polygon's shadowed and lit fragments are computed by filtering it down the shadow volume BSP tree. The polygonal scene with its computed shadows can be rendered with any polygon-based visible-surface algorithm. Since the shadow volumes and shadows are computed in object space, they can be used for further analysis of the scene. Pseudocode is provided, along with pictures and timings from an interactive implementation.

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—*Curve, surface, solid, and object representations*; I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Color, shading, shadowing, and texture*

General Terms: Algorithms

Additional Keywords and Phrases: shadows, shadow volumes, BSP, binary space partitioning

1 Introduction

One classic problem in 3D computer graphics is that of shadow generation. Areas in shadow are those that are not visible from a light source. The presence of shadows in an image helps viewers to better understand the spatial relationships between objects, is vital for applications such as architectural planning, and, in general, increases the appearance of reality that a picture provides. Unfortunately, current shadow generation algorithms do not run fast enough for interactive performance, except on special hardware [12]. Real-time alternatives to full shadow generation typically involve tricks for transforming polygons to create polygon shadows that are mapped onto one or more infinite planes [4]. These "fake" shadows are not properly clipped to the surfaces that they shadow and are not blocked by intervening surfaces.

We present a shadow algorithm that achieves interactive performance for polygonal environments of modest size when implemented on a graphics system that provides fast polygon scan conversion. After reviewing current shadow algorithms, we describe how the new algorithm is related to them. Next, we provide an overview of previous work on the BSP tree data structure and algorithms on which the shadow algorithm is based, and present a detailed description of how the new algorithm works.

2 Previous Shadow Algorithms

Crow's classic paper on shadow generation [8] describes three basic approaches: scanline shadow computation, the two-pass object-space approach, and shadow volumes. Since Crow's survey appeared, the taxonomy of shadow algorithms has been broadened to include three more basic methods: a two-pass z-buffer method [25], ray tracing [1, 24], and radiosity approaches [7, 17]. Because the algorithm discussed here combines the two-pass object-space approach with the shadow-volume approach, we provide a brief introduction to both.

The two-pass object-space approach, developed by Atherton, Weiler, and Greenberg [2] for arbitrary polygonal environments, applies two passes of an object-space visible-surface algorithm. The first pass, executed from the point of view of the light source, splits polygons into pieces that are visible from the light source (lit) and ones that are invisible from the light source (shadowed). This is accomplished by

transforming the polygons from the point of view of the light source and clipping those polygons that are further away against the clip window of those that are closer. Any part of a polygon that lies within a closer polygon, as seen from the light source, is in shadow. Lit polygon fragments are transformed back into their original orientation and attached to the original polygons as surface detail polygons. A second pass through the visible-surface algorithm is then performed from the point of view of the camera.

The shadow-volume approach involves the construction of a "shadow volume" for each object facing the light source. The shadow volume of an object is that volume bounded by the object and a set of invisible "shadow polygons," all of which face outward from the volume. A shadow polygon is created by connecting two vectors emanating from the point light source with the two vertices of one of the object's edges. The polygon is bounded by the edge, and the pieces of the two light source vectors that begin at the edge and continue away from the light source. The entire shadow volume is clipped against the view volume to yield a finite volume. Any part of a polygon within another polygon's shadow volume is shadowed. Whether a visible point on a scene polygon is in shadow can be determined by computing the relative number of shadow polygons between it and the eyepoint that are front-facing or back-facing. A number of shadow algorithms have been developed that create shadow volumes as a preprocessing step before rendering with a scan-line or z-buffer visible-surface algorithm [15, 5, 16, 3, 14, 9].

The technique described here combines elements of both these approaches, with some important differences [6]. In the two-pass object-space approach, the scene must be wholly within the light source's view volume and must be transformed by the light source's perspective transformation. While the algorithm described here does clip scene polygons into shadowed and lit parts, it does not require that polygons be transformed in the shadow generation process. The basis of the Atherton, Weiler, and Greenberg algorithm—the Weiler-Atherton polygon clipper [22] (or its more robust descendant [23])—contains a number of implementation subtleties. In contrast, our algorithm uses a simpler clipping algorithm that always clips a polygon against a plane, rather than against another polygon. The new algorithm's second (visible-surface) pass may be conveniently accomplished in image-space. Alternatively, the algorithm may also be used to perform object-space visible-surface determination by placing the light source at the eyepoint and returning the list of the non-overlapping lit (visible) polygon fragments that are computed.

Although the new algorithm generates a shadow volume, the volume does not have to be closed (e.g., by clipping it against the view volume) and does not include the actual scene polygons. Shadow-volume algorithms typically use the shadow volume to compute shadows in the course of performing visible-surface determination. The algorithm described here clips the scene polygons against the shadow volume in object space in the spirit of [16], creating the shadow volume as it proceeds.

Our algorithm benefits from the divide-and-conquer power of the Binary Space Partitioning (BSP) tree [10, 11] and its generalization to modeling polyhedra [20]. It is relatively simple and straightforward to implement and efficient enough to provide interactive performance. In order to understand how the algorithm works it is necessary to review some BSP fundamentals.

3 BSP Fundamentals

The BSP visible-surface algorithm, developed by Fuchs, Kedem, and Naylor, provides an extremely elegant and simple way to determine visibility priority among polygons in a scene independent of the eyepoint [10, 11]. A BSP tree represents a recursive partitioning of n-dimensional space, inspired by the early work of Schumacker [18, 19]. In 3D, the BSP tree's root is a polygon selected from those in the scene. The root polygon is used to partition object space into two half-spaces. One half-space contains all remaining polygons in front of the root polygon, relative to its plane equation, and the other contains all polygons behind it. Any polygon that lies on both sides of the root polygon's plane is split by the plane and its front and back pieces are assigned to the appropriate half-space. One polygon each from the root polygon's front and back half-space become its front and back children. Each child is recursively used to divide the remaining polygons in its half-space in the same fashion. The tree is complete when each leaf node contains only a single polygon whose two half-spaces are empty. A modified inorder traversal of this tree provides for $O(n)$ back-to-front ordering from an arbitrary viewpoint.

Thibault and Naylor [20] introduced the concept of using a BSP tree to represent polyhedral solids. They associate an "in" or "out" value with each empty region at the leaves. Assuming that a polyhedron's normals point outward, then an "in" region corresponds to the half-space on a polygon's back side, and an "out" region corresponds to the half-space on a polygon's front side. Each internal node defines a plane and has a list of polygons embedded in the plane. The "in" and "out" regions form a convex polyhedral tessellation of space. Thus, a BSP tree can represent an arbitrary (possibly concave) solid with holes as a union of convex "in" regions. Thibault and Naylor show how to produce a BSP tree from a polygonal boundary representation of a solid and how to perform Boolean set operations on two boundary representations or on a BSP tree and a boundary representation to yield a new BSP tree.

4 The SVBSP Algorithm

The Shadow Volume BSP (SVBSP) tree is a modified version of the BSP tree used by Thibault and Naylor. Each internal node is associated with a "shadow plane" defined by a point light source and an edge of a polygon facing the light source. (If a directional light is used, then the shadow plane is defined by the light source's direction vector and the polygon edge.) This is one of the "shadow polygons" that Crow refers to as bounding the shadow volume [8]. The direction of the plane's normal is used to determine the half-space in which an object is located. At the leaves are the "in" and "out" cells indicating whether or not a region is interior to the shadow volume.

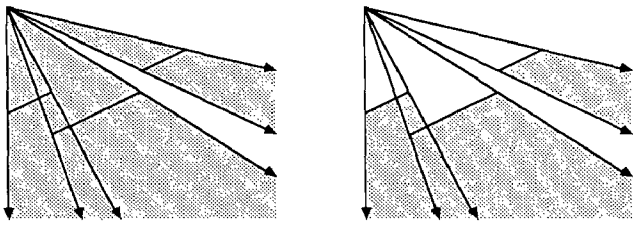
There are two basic steps to the Shadow Volume BSP algorithm whose execution is interleaved for each polygon facing the light source:

- *Determining shadows.* The polygon is filtered down the SVBSP tree to determine those parts that are shadowed and those that are lit.
- *Enlarging the SVBSP tree.* The shadow volume for each of the polygon's lit parts is created and added to the SVBSP tree.

The most straightforward approach to checking whether a polygon is in shadow would be to compare it with the shadow volumes of all other polygons. Polygons that are further from the light source than the polygon being tested cannot, however, cast a shadow on it. Therefore, if we process the polygons in front-to-back order relative to the light source, then each polygon would only have to be compared with the shadow volumes of those polygons that have already been processed and which are closer to the light source than it. The front-to-back ordering can be determined by building a regular BSP tree from the original scene polygons and traversing it from the point of view of the light source. Note, that this BSP tree needs to be created only once at the outset. It must be recomputed only if the scene's polygons change, not if a light source is moved.

Rather than check if a polygon is in each of the individual shadow volumes of all polygons in front of it, it is more efficient to keep one current merged shadow volume that is enlarged by unioning it with the shadow volume of each new polygon as the polygons are processed in front-to-back order. Since a polygon is compared only with polygons that are closer to the light than it, there is no need to check it against those planes of the merged shadow volume that would have been defined by these closer polygons. Therefore, these planes may be left out. (If complete shadow volumes are needed for subsequent computation, however, the scene polygon planes must be included.) The resulting merged shadow volume is a set of semi-infinite pyramids radiating outward from a single apex at the point light source. Figure 1 (a) shows a merged SVBSP shadow volume in 2D for a set of lines seen from a point, while Figure 1 (b) shows the shadow volume with the lines (planes in 3D) actually included. While Nishita et al. [16] compare the shadow volumes of two convex polyhedra, BSP trees make it relatively easy to compare the shadow volume of one polygon with a typically concave union of shadow volumes. The union operation is a version of the Boolean set union for polyhedra described in [20].

The determination of which areas of a polygon are in shadow is performed by filtering the polygon down the SVBSP tree,



(a) (b)

Figure 1 SVBSP volume in 2D.

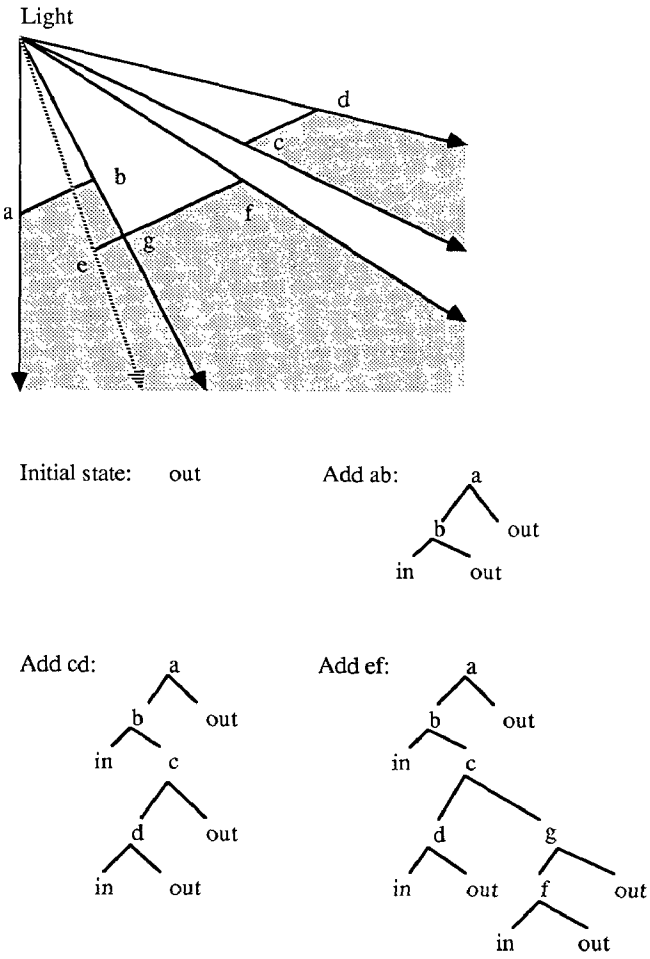


Figure 2 Building an SVBSP volume in 2D.

splitting it whenever it lies in both half-spaces of a node's plane. Fragments that reach the "in" leaves are in shadow, while fragments that reach the "out" leaves are lit. Since the SVBSP tree is built incrementally, each polygon is compared only with that part of the tree in existence when it is processed. Note that it is not necessary to filter any polygon that doesn't face the light source, since it is already entirely in shadow. Such polygons include any polygon whose plane embeds the light source.

The SVBSP tree must be augmented to include each lit fragment's shadow volume. This is accomplished by creating a set of shadow planes for the fragment's edges and constructing an SVBSP tree for them, using the algorithm for building BSP trees presented in [20]. An SVBSP tree node consists of the shadow plane alone, since the shadow plane edge is no longer needed. Figure 2 shows the steps in the construction of an SVBSP tree in 2D. Initially, the tree contains a single "out" cell. Lines *ab* and *cd* (which would be polygons in 3D) are both filtered down the SVBSP tree without any splitting. When line *ef* is filtered down the tree, it is split into *eg* and *gf* by the shadow plane through *b*. Line *eg* is wholly inside an "in" cell (the left branch of *b* in Figure 2);

therefore its shadow planes are not inserted. Since each fragment that reaches an "out" cell is lit, it casts a shadow that might fall on fragments added later. Therefore, shadow planes for each edge of a lit polygon fragment are computed and the volume that they define is added to the SVBSP tree, replacing the "out" cell in which the fragment landed.

Figure 3 shows pseudocode for shadowGenerator, the top-level shadow generation loop for the scene polygons. Each scene polygon is processed by the recursive procedure determineShadow (Figure 4), which filters the polygon down the SVBSP tree, splits it when necessary, and augments the SVBSP tree with the shadow volumes of lit fragments. The pseudocode shown here assumes convex polygons and a single light source.

5 Multiple Light Sources

The SVBSP algorithm described above can be easily modified to generate shadows cast by multiple light sources. This can be accomplished by building a separate SVBSP tree for each light source. All processing for one light source is performed before considering the next. Therefore, only one SVBSP tree need be kept in memory at any time. Polygons are processed in front-to-back order with respect to the current light source. Each polygon fragment must keep track of the light sources by which it is lit. If a fragment falls into an SVBSP tree "out" cell, it is marked as lit. If it falls into an "in" cell, it is marked as shadowed. In both cases, the polygon fragment is attached to the regular BSP tree node of the unfragmented polygon with which it is associated. After all the polygons have been

```
; shadowGenerator determines shadow fragments that
; are attached to the appropriate node in the BSP
; tree for subsequent rendering. Alternatively,
; fragments could be written to a file.

procedure shadowGenerator (PLS, BSPtree)

    ; Initialize the SVBSP tree to an OUT cell

    SVBSPtree := OUT

    ; Process all polygons facing light source PLS in
    ; front-to-back order by BSP tree traversal in O(n).

    for each scene polygon p, in front-to-back order
        relative to PLS

        if p is facing PLS

            ; Determine areas of p that are shadowed.
            ; BSPnode is p's node in BSPtree

            SVBSPtree := determineShadow (p, SVBSPtree,
                PLS, BSPnode)
        else
            ; p is not facing PLS or PLS is in p's plane

            mark p as shadowed
        endif
    endfor

    discard SVBSPtree
endproc
```

Figure 3 Pseudocode for shadowGenerator.

```
; determineShadow filters p down SVBSPtree to
; determine shadowed fragments and reattaches shadowed
; fragments to BSPnode.

procedure determineShadow (p, SVBSPnode, PLS, BSPnode)
    returns SVBSPnode

    if (SVBSPnode is an IN cell)
        attach p to BSPnode as a shadowed fragment
    else if (SVBSPnode is an OUT cell)
        attach p to BSPnode as a lit fragment

        ; create shadow volume for p and
        ; append it to the SVBSP tree

        shadowPlanes := planes that form the shadow
            volume of p with PLS
        SVBSPnode :=
            buildSVBSPtree (SVBSPnode, shadowPlanes)
    else
        ; Split p by SVBSPnode.plane, creating
        ; negPart and posPart.

        splitPolygon (p, SVBSPnode.plane, negPart, posPart)

        if (negPart is not null)
            SVBSPnode.negNode :=
                determineShadow (negPart,
                SVBSPnode.negNode, PLS, BSPnode)
        if (posPart is not null)
            SVBSPnode.posNode :=
                determineShadow (posPart,
                SVBSPnode.posNode, PLS, BSPnode)
    endif
endproc
```

Figure 4 Pseudocode for determineShadow.

processed in front-to-back order with respect to the current light source, the current SVBSP tree can be discarded and a new one initialized. The polygon fragments created by filtering the scene through the previous SVBSP tree are filtered through the next SVBSP tree in front-to-back order relative to the new light source. Note that the front-to-back order is established by traversing the original BSP tree, which has not gained any nodes due to SVBSP polygon fragmentation.

Shading calculations can be done after the entire scene has been processed for each light source, since each polygon fragment that has passed through the last light source's SVBSP tree is now associated with information indicating which light sources illuminate it. This is ideal for graphics systems that offer hardware shading support for multiple light sources. Alternatively, shading calculations could be performed incrementally as each light source's visibility from a fragment is determined.

6 Discussion

It is highly desirable to keep an SVBSP tree well-balanced, even at the expense of increased size, as is the case when using BSP trees to model polyhedra. This would help in unioning and filtering since each polygon must be filtered down to the tree's leaves. Controlling tree size is also important, however. One major way to accomplish this is to consider only the silhouette edges of the objects of which the polygons are part—a standard shadow algorithm optimization. As well, the edges created by splitting a polygon as it is filtered down the

original BSP tree need not be counted. Another way to reduce the size of the tree is to create shadow planes only for polygons that the user marks as being able to cast shadows.

In the special case of one light source, the shadowed fragments may be kept and the lit fragments thrown away, rather than keeping both. In this case, a polygon would be rendered by drawing its shadowed fragments on top of the original unfragmented polygon, as in [2]. There are some cases in which shadows may be known to fall only within a specified region, for example when a light source is defined with cones or flaps [21]. In these cases, the scene can first be clipped to a view volume containing only the region of interest for further processing. This could also be accomplished using a BSP tree.

The fragments produced by filtering down one light's SVBSP tree are pipelined to the next SVBSP tree. Therefore, in processing multiple point light sources, it better to proceed in the order that results in the least amount of polygon fragmentation. One heuristic is to process the light sources in increasing order of the number of polygons facing them. Alternatively, the light sources can be processed in increasing order of the likelihood with which their position will change. If copies of the intermediate fragments produced by each SVBSP tree for each polygon are maintained, then a change in the position of the i^{th} light source will only require sending the fragments from the $i-1^{th}$ SVBSP tree through the remaining trees. Therefore, those light sources whose position will change most often can be computed last.

Although a light source's SVBSP tree may be augmented with the shadow volumes of the lit polygon fragments that reach its leaves, these lit portions have already been fragmented by previous SVBSP trees. A smaller tree will result if the SVBSP tree is instead augmented by shadow volumes created from the more coherent lit fragments that result from filtering the original scene polygon down the current SVBSP tree alone. These more coherent fragments cannot be used for multiple light source rendering since they only record the effect of the current light source. They may, however, be used to render the effects of that light source by itself.

As Thibault and Naylor point out, fragmentation could also be reduced if edges were merged when it has been determined that adjacent fragments are both in "in" or "out" regions. As a special case, if all fragments of the polygon are lit or all are shadowed, then the fragments may be discarded and a copy of the original polygon used, marked accordingly.

7 Implementation

This algorithm has been implemented in C on a HP 9000 350 TurboSRX graphics workstation under HP-UX using the Starbase Graphics Library. To simplify the implementation only convex polygons are handled and polygons are processed individually, so no advantage is taken of the connectivity of polygons in polyhedra to identify silhouette edges. As well, no distinction is made between the original edges of a polygon and those generated by splitting it during creation of the original BSP tree or the SVBSP trees. A bit mask is used to keep track of which light sources illuminate each polygon fragment. Our implementation is able to take advantage of the hardware shading capabilities provided by the graphics system when rendering the figures. Since the original scene is already represented as a BSP tree, the scene may be rendered with either the BSP visible-surface algorithm (as done in the figures included here) or the hardware z-buffer. Timings for the figures are presented in Table 1 and include only the time needed to generate polygon shadows. Rendering time was an additional fraction of a second.

Figures 5–8 show a scene illuminated by three light sources, shown individually and together. Two versions of the scene are shown in each figure. The first version is shaded using the light sources. The second, fragmented version shows how the scene polygons are split by both the scene BSP tree and the light source SVBSP trees: shadowed fragments are shown in three levels of grey, depending on the number of light sources that illuminate them, while colored fragments are lit by all light sources. Figure 9 shows another scene with only the shadowed fragments outlined. Figures 10 and 11 show additional scenes rendered with the algorithm.

It is important to note that care must be taken to avoid problems posed by limited floating point precision. For example, as polygon fragments get progressively smaller due to fragmentation, the plane equation that would be calculated for each will also get progressively more inaccurate. We currently compute a plane equation for each original scene polygon and assign it to each polygon fragment generated from it. This not only saves computation, but assures that all fragments of the original polygon remain coplanar with each other. A similar approach can preserve the collinearity of edges that are formed by splitting an original scene edge.

8 Conclusions and Future Work

The algorithm that we have presented generates shadows in object space in near real-time for a modest number of polygons. It is simple to implement, and because it generates a set of polygons as output, it may be used as a preprocess to any polygon-based visible-surface algorithm. Since the input and output formats are the same, a pipelined approach for modeling shadows from multiple light sources is easy to implement. No restrictions are placed on the locations of the point light sources and viewer; no transformations are required before visible-surface determination. Our implementation relies on the use of a BSP tree representation of the scene to determine a front-to-back ordering of the scene polygons for each light source in time linear in the number of scene polygons. The algorithm may be easily modified to support a full shadow volume that includes the scene polygons, in which case the scene BSP tree is not necessary. We have recently learned that Naylor (personal communication, 1989) has independently proposed a similar algorithm.

BSP trees not only present a unified framework for visible-surface determination, point classification, and set operations on polyhedra, but, as we have shown, also make possible interactive shadow generation on modern graphics workstations. In addition to pursuing some of the performance improvements mentioned previously, we are also investigating a natural extension to the SVBSP algorithm to support object-space shadow generation for linear and area light sources [6].

Figure	Secs	Lights	Input polygons	Front-facing polygons	Front-facing edges	SVBSP nodes	Fragments
5	.62	1	27	12	49	144	122
6	.68	1	27	15	61	140	108
7	.23	1	27	15	61	31	49
8	5.33	3	27	12,15,15	49,61,61	144,140,32	475
9	1.96	1	126	61	227	88	537
10	4.97	2	65	33,32	132,128	210,169	523
11	7.99	2	106	49,53	196,212	415,191	813
Tetra256	4.15	1	258	130	390	577	723
Tetra1024	25.44	1	1026	514	1542	2894	3345

Table 1 Timings for figures. Figures Tetra256 and Tetra1024 (not shown) are recursive tetrahedra [13] with 256 and 1024 polygons, respectively, casting shadows on themselves and a ground plane.

Acknowledgements

This work is supported in part by the Defense Advanced Research Projects Agency under Contract N00039-84-C-0165, an equipment grant from the Hewlett-Packard Company AI University Grants Program, and the New York State Center for Advanced Technology under Contract NYSSTF-CAT(88)-5. The recursive tetrahedron in Figure 9 was generated using Eric Haines's Standard Procedural Database [13].

References

1. Appel, A. "Some Techniques for Shading Machine Renderings of Solids." *AFIPS SJCC 68*, 32, 1968, 37-45.
2. Atherton, P., Weiler, K., and Greenberg, D. "Polygon Shadow Generation." *Proc. SIGGRAPH '78*. In *Computer Graphics*, 12:3, July 1978, 275-281.
3. Bergeron, P. "A General Version of Crow's Shadow Volumes." *IEEE CG&A*, 6:9, September 1986, 17-28.
4. Blinn, J. "Jim Blinn's Corner: Me and My (Fake) Shadow." *IEEE CG&A*, 8:1, January 1988, 82-86.
5. Brotman, L.S., and Badler, N. "Generating Soft Shadows with a Depth Buffer Algorithm." *IEEE CG&A*, 4:10, October 1984, 5-12.
6. Chin, N. "Shadow Generation Using BSP Trees." M.S. Thesis, Dept. of Computer Science, Columbia University New York (forthcoming), 1989.
7. Cohen, M. and Greenberg, D. "The Hemi-Cube: A Radiosity Solution for Complex Environments." *Proc. SIGGRAPH '85*. In *Computer Graphics*, 19:3, July 1985, 31-40.
8. Crow, F. "Shadow Algorithms for Computer Graphics." *Proc. SIGGRAPH '77*. In *Computer Graphics*, 11:3, July 1977, 242-248.
9. Fournier, A. and Fussell, D. "On the Power of the Frame Buffer." *ACM Trans. on Graphics*, 7:2, April 1988, 103-128.
10. Fuchs, H., Kedem, A., and Naylor, B. "On Visible Surface Generation by A Priori Tree Structures." *Proc. SIGGRAPH '80*. In *Computer Graphics*, 14:3, July 1980, 124-133.
11. Fuchs, H., Abram, G., and Grant, E. "Near Real-Time Shaded Display of Rigid Objects." *Proc. SIGGRAPH '83*. In *Computer Graphics*, 17:3, July 1983, 65-72.
12. Fuchs, H., Goldfeather, J., Hultquist, J., Spach, S., Austin, J., Brooks, Jr., F., Eyles, J., and Poulton, J. "Fast Spheres, Shadows, Textures, Transparencies, and Image Enhancements in Pixel-Planes." *Proc. SIGGRAPH '85*. In *Computer Graphics*, 19:3, July 1985, 111-120.
13. Haines, E. "A Proposal for Standard Graphics Environments." *IEEE CG&A*, 7:11, November 1987, 3-5.
14. Max, N. "Atmospheric Illumination and Shadows." *Proc. SIGGRAPH '86*. In *Computer Graphics*, 20:4, August 1986, 117-124.
15. Nishita, T. and Nakamae, E. "Half-Tone Representation of 3-D Objects Illuminated by Area Sources or Polyhedron Sources." IEEE COMPSAC, November 1983, 237-242.
16. Nishita, T., Okamura, I., Nakamae, E., "Shading Models for Point and Linear Light Sources." *ACM Trans. on Graphics*, 4:2, April 1985, 124-146.
17. Nishita, T. and Nakamae, E. "Continuous Tone Representation of Three-Dimensional Objects Taking Account of Shadows and Interreflection." *Proc. SIGGRAPH '85*. In *Computer Graphics*, 19:3, July 1985, 23-30.
18. Schumacker, R., Brand, B., Gilliland, M., and Sharp, W. "Study for Applying Computer-Generated Images to Visual Simulation." AFHRL-TR-69-14, USAF Human Resources laboratory, September 1969.
19. Sutherland, I., Sproull, R., and Schumacker, R. "A Characterization of Ten Hidden-Surface Algorithms." *ACM Comp. Surv.*, 6:1, March 1974, 1-55.
20. Thibault, W. and Naylor, B. "Set Operations on Polyhedra Using Binary Space Partitioning Trees." *Proc. SIGGRAPH '87*. In *Computer Graphics*, 21:4, July 1987, 153-162.
21. Warn, D. "Lighting Controls for Synthetic Images." *Proc. SIGGRAPH '83*. In *Computer Graphics*, 17:3, July 1983, 13-21.
22. Weiler, K. and Atherton, P. "Hidden Surface Removal Using Polygon Area Sorting." *Proc. SIGGRAPH '77*. In *Computer Graphics*, 11:2, July 1977, 214-222.
23. Weiler, K. "Polygon Comparison using a Graph Representation." *Proc. SIGGRAPH '80*. In *Computer Graphics*, 14:3, July 1980, 10-18.
24. Whitted, T. "An Improved Illumination Model for Shaded Display." *CACM*, 23:6, June 1980, 343-349.
25. Williams, L. "Casting Curved Shadows on Curved Surfaces." *Proc. SIGGRAPH '78*. In *Computer Graphics*, 12:3, August 1978, 270-274.

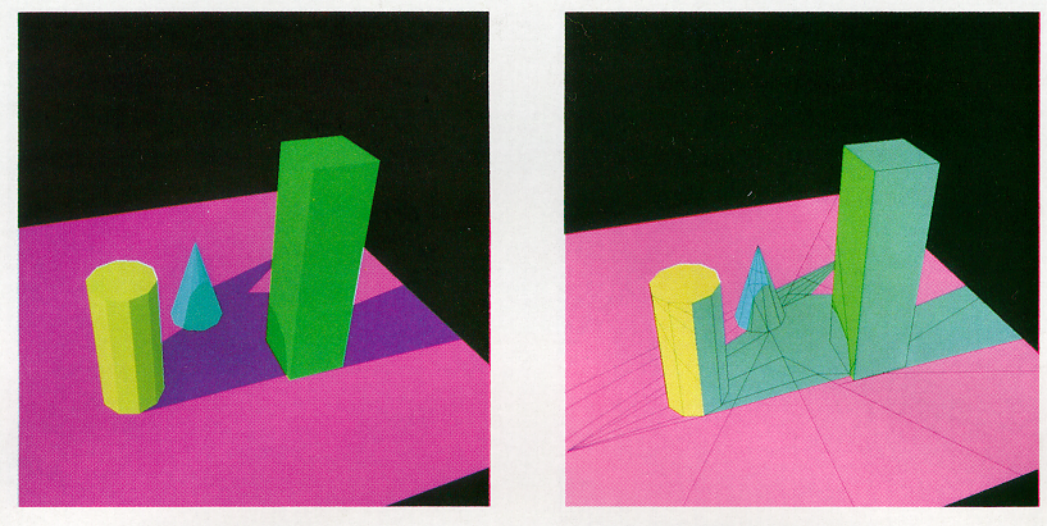


Figure 5 Solids with light source 1.

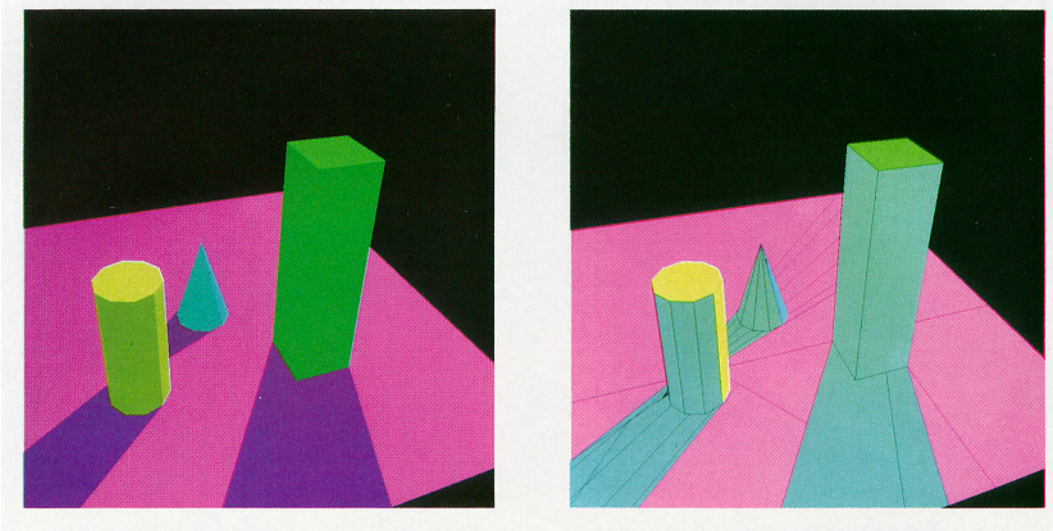


Figure 6 Solids with light source 2.

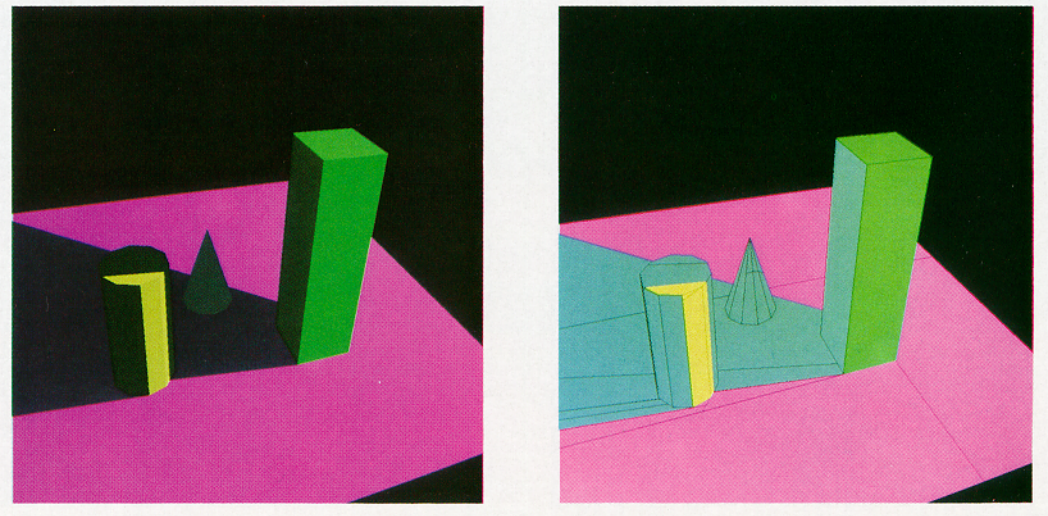


Figure 7 Solids with light source 3.

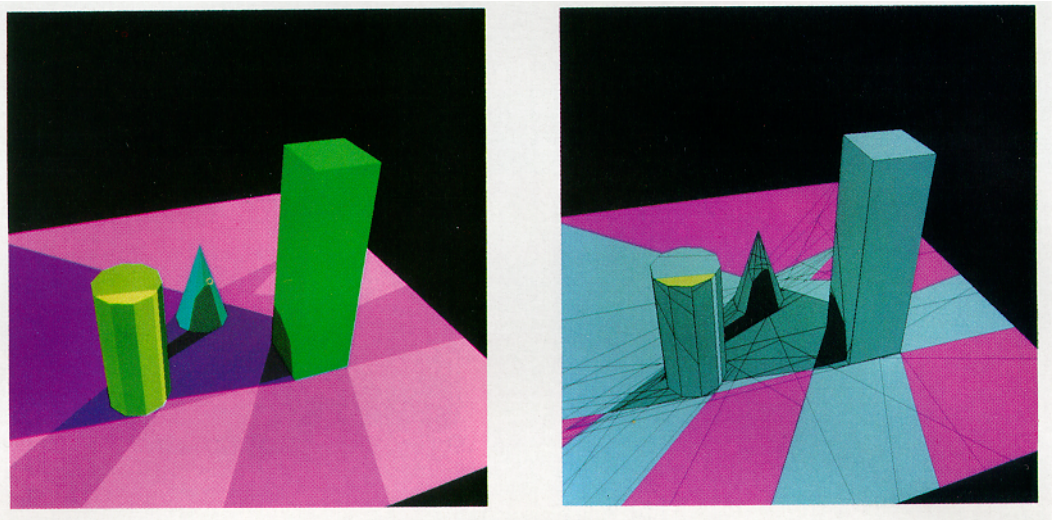


Figure 8 Solids with all three light sources.

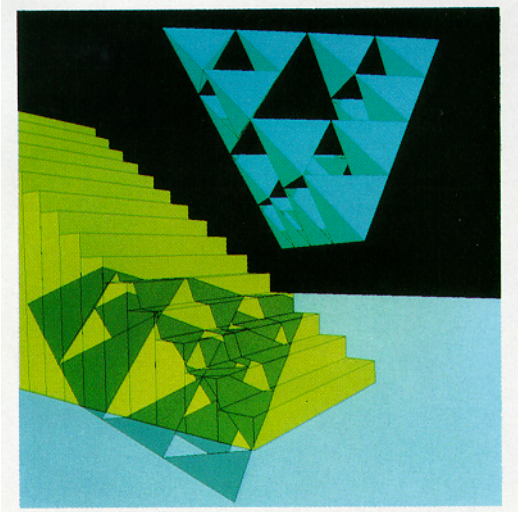


Figure 9 Recursive tetrahedron and staircase with one light source, showing shadow fragments.

Figure 10 Table and chair with two light sources.

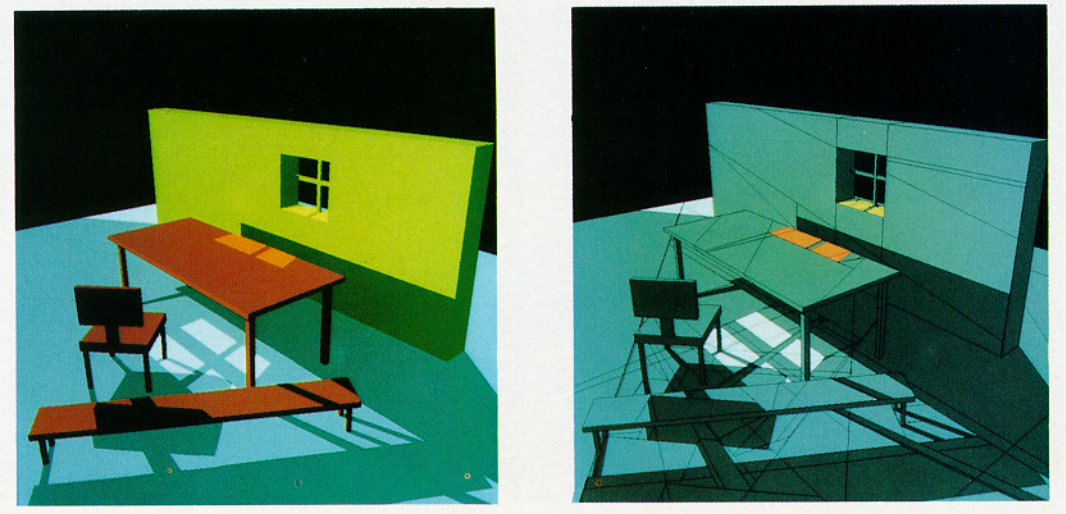


Figure 11 Room with two light sources.