Computing the Boundary of a Class of Labeled-Leaf BSP Solids

Sherif Ghali* Chris Smith*

Abstract

We describe an algorithm that computes the boundary of the shadow volume cast by a collection of piecewise linear polyhedra in space using BSP trees. Unlike boundary representations, representing solids in general and shadow volumes in particular using BSP trees makes it possible to implement boolean operations easily and robustly. Also, in contrast with operating in Constructive Solid Geometry or on Nef polyhedra, no neighborhood analysis is needed.

1 Introduction

Shadow computation is an important problem in computer graphics. The relative depth of objects often cannot be perceived by an observer unless shadows are rendered. One popular method for computing shadows is the shadow volume [5]. It has traditionally only been used with ray tracing to produce accurate shadows, but the availability of stencil buffers, a per-pixel counter, made shadow volumes usable when rendering using scanline algorithms as well [7].

The boundary of the shadow volume of one polyhedron consists of the polyhedron's polygons that face the light, the projection of these polygons (after reversing their orientation) on the ideal plane, and quadrangular polygons joining the affine silhouette edges of the polyhedron to the corresponding ideal edges. The polygons defining the boundary of the shadow volume are termed *shadow polygons*.

A point in space is in shadow if it is inside one shadow volume. If the shadow volumes of more than one polyhedron overlap, the boundary of the combined volume does not need to be computed. It suffices instead to count the number of independent shadow volumes a point is in. If the number is one or more, the point is in shadow.

But minimizing the combinatorial and the geometric size of the shadow volumes reduces the rendering time. In scanline rendering in particular, a reduction in the total number of pixels that all shadow polygons rasterize to will reduce the rendering time. An example in 2D is shown in Figure 1.

We describe an algorithm that computes the boundary of the union of the shadow volumes. We cannot

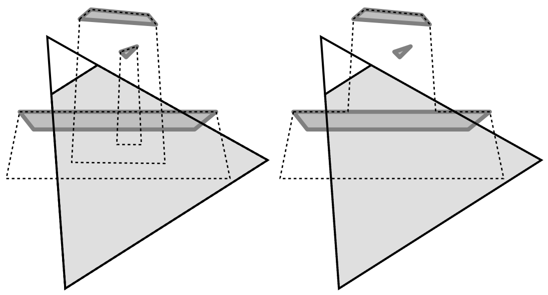

Figure 1: The three shadow volumes shown dashed on the left are merged into the single volume shown on the right – the shadow volumes extend to infinity, but are shown finite in size for clarity. By reducing the geometric size of the polygons describing the boundary of the shadow volume the polygons inside the view frustum are minimized, which optimizes shadow rendering time. The view frustum is shown as a finite triangle although it also is infinite.

hope to perform this computation at every frame, but if a scene consists of a combination of static and dynamic polyhedra, and if, as is often the case, the number of static polyhedra is large compared to the dynamic ones, then it is sensible to preprocess the static polyhedra and compute the union of their individual shadow volumes. The shadow volume of the dynamic polyhedra can still be generated independently.

2 Boolean Operations on Polyhedra

If two polyhedra `A` and `B` are described using their boundary `Abrep` and `Bbrep`, then `Cbrep = Abrep op Bbrep`, the boundary of the polyhedron `C` resulting from applying the operator `op` on `A` and `B`, can be determined by computing and classifying the pairwise intersection of faces from `A` and `B` [8]. Although such algorithms are at the outset simple, an implementation is usually quite intricate since neighborhoods must be maintained and that gives rise to many special cases.

An alternative representation of solids, Constructive Solid Geometry, makes it easy to perform boolean operations, but a conversion to boundary representation remains necessary since many algorithms, such as those for visualization, require boundaries as input. The conversion in turn requires performing boolean operations

*Department of Computing Science, University of Alberta, {ghali,csmith}@cs.ualberta.ca

on the boundary of the primitive leaf nodes.

As with CSG, Nef polyhedra [1], the point sets represented by constructing boolean expressions on (open) halfspaces, make it easy to perform set operations. Nef polyhedra have the additional advantage that they define a closed algebra. The point sets they capture include dangling edges and faces as well as non-manifold solids.

Although the first application of Binary Space Partitioning trees was the computation of depth orders [6], labeled-leaf BSP trees can be used to represent solids and to perform boolean operations [11, 10]. In this representation each interior node of the tree stores an oriented plane that divides space into two open halfspaces corresponding to its two children. Each leaf node represents the open convex polyhedron defined by the intersection of the corresponding halfspaces of its ancestors. By storing a flag 'in' or 'out' at each leaf node, a polyhedron can be represented as the closure of the convex polyhedra at the leaves marked 'in'.

An algorithm to determine the boundary of a labeled-leaf BSP tree has been described [4], but that algorithm first constructs the boundary of the convex polyhedra at the leaves and then merges the resulting polyhedra. The algorithm also appears to be too involved to be practical.

We describe the first practical algorithm to compute the boundary of a BSP tree representing a shadow volume. Because all visibility events occur along planes defined by silhouette edges and the light, only one-dimensional occlusion needs to be handled. Another way of looking at it is that although each leaf of the labeled-leaf BSP tree represents a convex unbounded polyhedron, all splitting planes pass by the light. The partition of space is thus a partition of a sphere centered at the light. Rather than introduce unnecessary nodes in the tree storing the carrying planes of polygons visible from the light, each such polygon is stored in the tree at an 'out' node. For this procedure to work, the polygons need to be processed in front-to-back order from the light using a secondary BSP tree. This crucial observation has been made by the authors of an algorithm that partitions a set of polygons into those visible and those invisible from a light source [3].

3 The Boundary of the Set Union of Shadow Volumes

We assume that the polygons defining each polyhedron in the input are convex, though each polyhedron does not need to be convex. A set of polygons defining the boundary of each input polyhedron is extracted while saving for each edge of each polygon whether it is a silhouette edge as seen from the light. The secondary BSP tree is used to compute a list $\mathcal{L}$ of the polygons in front-to-back order with respect to the light.

Each node in the labeled-leaf BSP tree $\mathcal{C}$ is defined by the plane passing through the light source O and an edge on one of the polyhedra. The plane is defined such that its normal vector points outside the shadow volume. Each node stores the following attributes:

- An (oriented) splitting plane.
- A label {Interior, 'in', 'out'}.
- An edge e that, alongside the light, defines the splitting plane.
- A flag identifying whether that edge is a silhouette edge.
- For each edge e that is a silhouette edge, a set of intervals $\mathcal{I}$ is maintained. The intervals identify the set of points P along the edge such that the ray OP does not intersect any of the polygons already inserted in the tree $\mathcal{C}$. An example of the intervals is shown in Figure 2.
- If the node is labeled 'out', the polygon visible from the light.
- If the node is labeled Interior, two pointers to its children nodes.

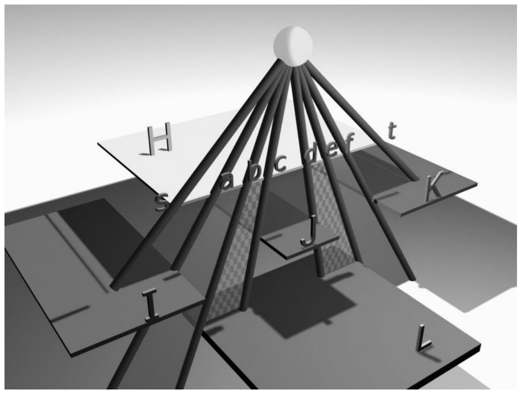

Figure 2: The polygon L has just been partitioned by the front edge of polygon H. $\mathcal{I}$ at that edge is $\{[a,c],[d,f]\}$. β is $\{[b,e]\}$. The intersection yields $\{[b,c],[d,e]\}$ and describes the two cross-hatched polygons. The difference is $\{[a,b],[e,f]\}$ and is assigned back to $\mathcal{I}$.

The labeled-leaf BSP tree $\mathcal{C}$ is initialized to a single node labeled 'out' identifying that the shadow volume is initially empty. The polygons in $\mathcal{L}$ are processed in order. Each polygon is extracted and inserted into the labeled-leaf tree. The processing of a polygon depends on the type of node reached:

Interior The following actions are taken:

1. The polygon is divided by the splitting plane at the node and the segment of intersection is saved.
2. The intersection segment is projected through O on the edge e and the resulting interval β along e is noted.
3. Two one-dimensional set operations are performed. The difference $\mathcal{I} - \beta$ and the intersection $\mathcal{I} \cap \beta$ are determined. The difference describes the set of points along e that, projected on the polygon being partitioned, describes one or more polygons of the shadow volume. The intersection describes the set of points that remain uncovered. It is assigned to $\mathcal{I}$ at the current node. See Figure 2.
4. The two polygon fragments resulting from the split are inserted recursively at the two child nodes.

'in' The polygon is not visible from the light, no action is taken.

'out' The polygon is visible from the light. Each of its edges is combined with the light source to construct a new set of splitting planes. These splitting planes are used to build a new subtree that is attached at the node currently labeled 'out'. See Figure 3.

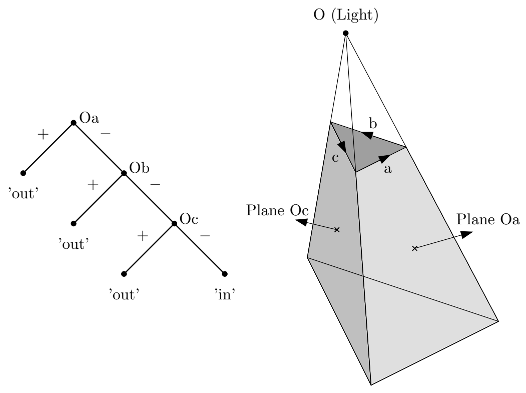

Figure 3: When the polygon *abc* reaches an 'out' node, the node is replaced by the subtree shown on the left.

After the polygons in $\mathcal{L}$ have been inserted, an additional traversal of $\mathcal{C}$ visits all nodes at which $\mathcal{I}$ is not empty. For each interval remaining in a set $\mathcal{I}$, a quadrangular polygon with two affine and two ideal vertices is generated.

4 Implementation

A video demonstrating our implementation can be downloaded from `http://www.cs.ualberta.-ca/~ghali/papers/cccg05/brep-BSPsolid.mp4`. It can be viewed using the Apple Quicktime viewer. Building the implementation on CGAL made it possible to experiment with different number types. As would be expected, floating point computation leads to incorrect results or to crashes. For the largest scene we tried, consisting of 10000 polygons, we were able to use floating point arithmetic in combination with an ad-hoc epsilon value. This value is used as a clamp when performing polygon binary split operations: Vertices closer to the splitting plane than this epsilon are considered to lie on the plane. In general, however, it is not possible to find an epsilon that allows us to use floating point arithmetic on an arbitrary input. Using exact arithmetic and handling special (or degenerate) cases as "first-class citizens" [2] allows us to handle arbitrarily large inputs.

In addition to reducing the average rendering time, preprocessing a scene by computing the reduced-size shadow polygons is useful to reduce the large fluctuation in rendering time that otherwise occurs. Such a large fluctuation occurs when many shadow polygons enter then exit the field of view. Figure 4 shows a comparison between the per-frame rendering times of the same animation.

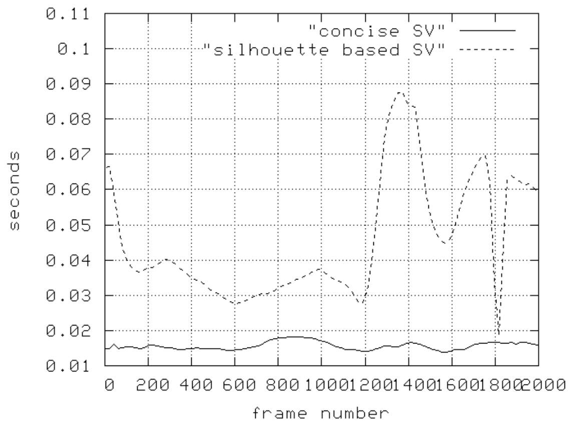

Figure 4: Comparison between the time it took to render each frame in an animation of the scene shown in Figure 5. Dashed line: a separate shadow volume is used for each polyhedron. Solid line: the polygons defining the boundary of the union of the shadow volumes are used.

5 Future Work

Several representations of solids are known: boundary representations, BSP trees, CSG, Nef polyhedra, and spatial enumeration. Modeling systems have long needed to use more than one representation for solids because some operations are easier to perform on one representation than another. This is also the reason we use labeled-leaf BSP trees rather than a boundary rep-

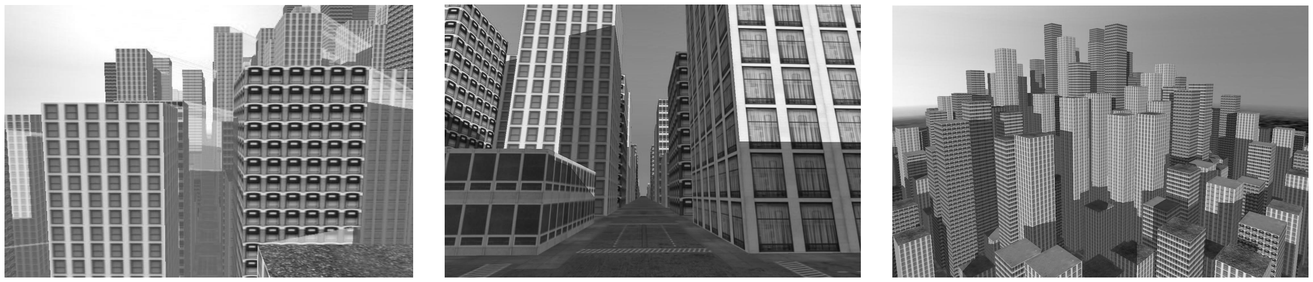
(1) (2) (3)

Figure 5: (1) The slanted polygon are the boundary of the BSP solid. Computing the boundary optimizes shadow rendering, which is shown when the viewer is either inside (2) or outside (3) the shadow volume.

resentation. The method we describe for generating a boundary representation from a BSP solid relies on the solids being shadow volumes. The main question that needs investigation is whether an efficient algorithm can be developed for computing the boundary of a general labeled-leaf BSP polyhedron. In effect our method only needs a 1D BSP tree in addition to a 3D BSP tree to compute the boundary because the visibility only matters along lines passing by the light source. A solution to the general problem would require maintaining 1D as well as 2D BSP trees to describe the facets of the BSP solid. Comba and Naylor's topological BSP trees [4], whereby the tree is threaded with cross-links between lower dimensional features, is a step towards solving the problem, but an algorithm that does not first generate the description of the boundary of individual leaves only to remove them later would be more interesting.

Another intriguing problem is the generation of analytical visibility with adjacency. How can one generate a Doubly-Connected Edge List that represents the view from an observer and that partitions a small sphere centered at the observer into faces corresponding to the view? BSP trees apear to be a promising method for solving this problem. It is most interesting in this case to operate directly on a sphere. It is not necessary to declare a view plane and to apply a perspective transformation [9]. This is more of a software engineering problem than a geometric one. The difficulty of manipulating DCELs means that a visibility algorithm must construct a DCEL using Euler operators [8].

6 Conclusion

We described the first practical algorithm to compute the boundary of a solid represented by a BSP tree. This makes it possible to compute the boundary of the union of a set of solids following a well-known technique [11]. Our algorithm relies on the solids being cast from a single source such as those resulting during shadow computation, which makes it possible to compute the boundary using only one and three dimensional data structures for point sets. Finding a practical algorithm that uses one, two, and three dimensional structures to compute the b-rep of an arbitrary BSP solid is an intriguing open problem.

References

[1] H. Bieri and W. Nef. Elementary set operations with d-dimensional polyhedra. In *Computational Geometry and its Applications*, volume 333 of *Lecture Notes Comput. Sci.*, pages 97–112. Springer-Verlag, 1988.

[2] C. Burnikel, K. Mehlhorn, and S. Schirra. On degeneracy in geometric computations. In *Proc. 5th ACM-SIAM Sympos. Discrete Algorithms*, pages 16–23, 1994.

[3] N. Chin and S. Feiner. Near real–time shadow generation using BSP trees. In *SIGGRAPH '89*, pages 99–106, Aug. 1989.

[4] J. Comba and B. Naylor. Conversion of binary space partitioning trees to boundary representation. In *Proceedings of Theory and Practice of Geometric Modeling*, Tuebingen, Germany, Oct 1996.

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

[6] H. Fuchs, Z. M. Kedem, and B. Naylor. On visible surface generation by a priori tree structures. *Computer Graphics*, 14(3):124–133, 1980. Proc. SIGGRAPH '80.

[7] T. Heidmann. Real shadows, real time. *Iris Universe*, 18:28–31, 1991. Silicon Graphics, Inc.

[8] M. Mäntylä. *An Introduction to Solid Modeling.* Computer Science Press, Rockville, MD, 1988.

[9] B. Naylor. Partitioning tree image representation and generation from 3d geometric models. In *Proceedings of Graphics Interface '92*, pages 201–212, may 1992.

[10] B. Naylor, J. A. Amanatides, and W. Thibault. Merging BSP trees yields polyhedral set operations. *Comput. Graph.*, 24(4):115–124, Aug. 1990. Proc. SIGGRAPH '90.

[11] W. C. Thibault and B. F. Naylor. Set operations on polyhedra using binary space partitioning trees. *Computer Graphics*, 21(4):153–162, 1987.