HIERARCHICAL APPROACHES TO HIDDEN SURFACE INTERSECTION TESTING

Norm Dadoun, David G. Kirkpatrick
University of British Columbia
John P. Walsh
Barron & Assoc. Acoustical Consultants Ltd.

ABSTRACT

An approximating hierarchical representation is presented for performing hidden surface elimination in complex scenes. Methods for constructing and using this hierarchy are discussed. Convex bodies are used as elements in this hierarchy. Recent results from computational geometry are applied to element construction and intersection processing.

KEYWORDS: computer graphics, computational geometry, hidden surface elimination, hierarchical structures, clipping.

1.0 Introduction

In this paper we discuss ways of structuring environment descriptions hierarchically to efficiently solve the hidden line/surface elimination problem. Our algorithms operate in object space [SSS,74] and can be used to operate on scenes of considerable complexity.

Our interest in this area did not come out of conventional work in computer graphics but arose from an ongoing research project involving computer simulation of concert hall acoustics [W,79] [W,80] [WD,81] [WD,82]. The simulation uses a geometric model of a room or concert hall which has been generated by a Computer Aided Architectural Design (CAAD) system [BEH,79]. Using this model, given a source position and a receiver position, we simulate the acoustics encountered at the receiver position. This is achieved by 'beam-tracing': a sound beam (modeled as a solid angle with a simple polygonal cross-section) emanates from the source position and is traced through the room (along with reflection, diffraction, absorption, and related effects) to determine whether or not the beam will impinge on the defined receiver position within a given trace length.

The underlying geometrical problem, testing intersections related to the beam tracing, can be regarded as a variant of the well-known hidden surface elimination problem with several additional constraints. A solution to the problem must answer the question: From a given source position, which surfaces in the room being modeled can be 'seen' within the limitations of this beam? The traditional hidden surface problem can be considered static with one viewer position and one viewing window. Our problem can be considered dynamic in that we have many viewing positions and many viewing windows. In our simulation, a single beam striking across several surfaces is 'split' into a number of reflections each of which must also be traced as a 'new' source position. With several hundred primary (pre-reflection) source beams and potentially exponential growth in the number of secondary (reflected) beams, our efforts have concentrated on segmenting and preprocessing the geometric model to facilitate the beam/surface testing.

The use of geometric hierarchies in structuring domain information for computer graphics was first suggested by Clark [C,76]. He proposed using

hierarchies to limit the amount of information under consideration at any particular time (the "working set") and also discussed generalizing the idea of clipping to include resolution clipping and other optical effects. He suggested that search algorithms take the form of a recursive descent through the hierarchy.

Independently, Rubin and Reddy [RR,78] developed the idea of a hierarchical representation for 3-dimensional objects for use in artificial intelligence applications. This was later refined and applied to computer graphics in a paper by Rubin and Whitted [RW,80]. They used a hierarchy of arbitrarily oriented parallelepipeds (referred to as bounding boxes in the following discussion) to segment their environments and used image space scan-line algorithms to render complex shaded scenes. Their hierarchy elements were created manually using a structure editor with no attempt to optimize systematically the volume or orientation of their bounding boxes.

Our work further explores the notion of a hierarchical representation by considering other candidates for hierarchy elements, examining the complexity of automatically segmenting the space and constructing the hierarchy elements, and applying recent results in the field of computational geometry to the construction, search and clipping processes. We have attempted to describe our results in as general a way as possible. A beam here can be considered a sound beam in our application or a viewing window in a typical graphics application.

2.0 Definitions

The two-dimensional convex hull of a planar set of points p is the unique minimum area convex polygon which contains p. The most natural representation is a circular list of vertices describing the outer contour of the hull. The three-dimensional convex hull of a set of points in three-space is the minimum volume convex polyhedron which contains that set of points. There are several options for representing the 3D hull. The simplest is a graph whose vertices are the extreme points of the convex set and whose edges denote adjacency on the convex hull. Convex hulls are approximations of objects which have nice computational properties due to their convexity. In recent years, a wealth of efficient algorithms for constructing and using these structures has been devised by researchers in computational geometry [DK,81] [MP,77] [PH,77] [Sh,78].

A bounding box is a cruder approximation with a simpler representation. Given a set of points p, a bounding box is the minimum volume arbitrarily oriented parallelepiped which contains p. It can be specified as a 4x4 transformation matrix (into local 'box' coordinates) and x, y, and z offsets to define the limit of the box. Rubin and Whitted used a related approximation element in their hierarchy although they did not attempt to minimize the volume of their 'boxes'. Construction and use of these convex elements is discussed in Section 4.

Our environment description is a boundary representation [R,80] which uses planar faces to describe the boundaries of the scene being modeled. We assume as input vertices, edges and planar faces, along with face adjacency information. Edges are specified as pairs of vertices. A face is specified as an edge ring from which the face plane equation can be derived.

Face adjacency information can be specified locally by having a pointer from each each edge to its two adjoining faces or globally as a graph relation by defining a dual graph with nodes relating to faces, and edges relating to adjacencies. The global graph relation of adjacency information can be derived from the edge information by using a depth-first search to extract the connected components.

Although object coherence is not required in the description of elements in the given environment, face orientation may be incorporated to describe the 'outside' of a wall or object. It may also be used to eliminate back faces (faces oriented away from the viewer) from consideration.

Within a geometric model of a scene as described above, a viewing window is defined as a viewing (source) position

and rays leaving this viewing position delimiting a beam. This beam could have an arbitrary polygonal cross-section (non-convex with holes). By definition, beams have contiguous cross-sections. A beam with disconnected components can be treated as several beams. The representation of a beam is the circular counter-clockwise list of rays forming the outside contour of the beam followed by a list of circular clockwise ray lists defining holes within the beam contour. A standard rectangular viewing screen would be modeled here as 4 rays leaving the viewer position and defining a truncated viewing pyramid.

Our definition of beam is extremely general. The beam could be regarded as defining an entire graphics viewing window which after hidden-line removal could used as input to a vector display. At the other extreme, the beam window could be regarded as a single pixel (picture element). Thus the extraction of values for all pixels could be used to generate a raster scan image.

3.0 Hierarchical Processing

Given a particular geometric model, we wish to be able to quickly identify all faces or portions of faces visible within any beam with arbitrary position and direction.

The brute force algorithm would test all faces exhaustively for containment within the beam window. A more intelligent approach might be to form rough clusters of faces and to test the beams against these clusters. The review paper by Sutherland et al explains and categorizes ten hidden surface elimination algorithms and general techniques for solving typical associated subproblems [SSS,74]. The uninitiated reader is directed to that survey.

Our desire is to trade search time for preprocessing time. Our approach is to use a hierarchical approximating structure as an alternate representation of the scene. This structure is organized so that questions about the original scene can be answered quickly by referring to this approximation. It is composed of polyhedral elements, each of which contains a portion of the structure to be approximated, arranged into a hierarchy. The root of this hierarchy is the largest element and contains the entire scene; the levels below the root are formed by splitting the scene into segments. The segments are further subdivided until the leaves are formed. Each leaf is a 2D convex approximation of one of the planar face elements making up the scene.

The processing involved in solving this problem must include the work required to massage the input data into an acceptable form. Thus a general outline might be:

1) take as input some unstructured scene representation.
2) extract from this initial representation the necessary plane equations and face adjacency information.
3) partition this representation into a segmented hierarchy.
4) replace each segment in the hierarchy with a containing polyhedral element.
5) input a viewing (source) position and delimiting rays defining a beam.
6) proceed with searching and clipping algorithms.

Items 1-4 refer to preprocessing the structure which is performed once; 5-6 refer to the hierarchical search which is to be performed many times. Items 1-4 are dealt with in the next subsection; items 5-6 are examined thereafter.

3.1 Preprocessing

What type of input data might a potential user have available to work with? On one hand, Rubin and Whitted assumed a "point or surface representation ... [not] aggregated into any hierarchical form" [RW,80,p.113]. At the other end of the spectrum, according to Walsh: "a hierarchy of walls (and/or wall elements), planar faces, [edges,] and vertices" [W,79,p.236]. This representation was intended to be the output of a CAAD system and thus already has much of the structure we need associated with it.

Obviously with this range of potential users, little in the way of predefined structure can be assumed. Different users may be able to step into the stream in various places according to how their data is initially structured but most users will require some 'massaging' of data into appropriate

forms. All of this could be done manually using structure editors as Rubin did. However, the potential tedium of such an exercise combined with the desire to optimize various aspects of the final structure leads us to isolate portions of the preprocessing and offer suggestions as to how to deal with each. Figure 1 identifies the appropriate portions and labels the necessary transformations.

With the multitude of representations in general use [R,80], the first transformation (a) may be into a geometric vertex boundary surface representation. Some of the associated object structure from some representations (for example, the primitive object descriptions in Constructive Solid Geometry) can be used to guide some of the hierarchical segmentation later on. In some cases, a non-planar surface may have to be approximated by planar faces.

Somehow, edge and face descriptions have to be specified and it seems unlikely that much of this could be automated if the data was in the form of digitizer input. The face adjacency information could initially be specified locally as edge/face relations and the dual adjacency graph could be derived from this by using a depth first search.

The extent of the next transformation depends on the complexity of the environment being structured. Transformation (b) involves the recursive partitioning of the faces and edges in the scene into segment groupings. The connected components of the adjacency graph can be used as a first slice into the segmentation. The segments can be further decomposed using various heuristics depending on the composition of the scene. Obviously, a cityscape description has a different fundamental description from a digital terrain model of a mountainous scene and its segmentation will use different heuristics.

The adjacency graph can also be used in several other ways. An articulation point in a graph is one which when removed causes the graph to become disconnected. These points in the adjacency graph may be used to identify objects sitting on single faces. In a similar way, high degree vertices can be used as "cluster points" for segmenting. This can be combined with schemes using edge angles, surface area weights, and/or statistical clustering of significant features. The segmentation can be driven by a reduction in the overall volume or the sizes of point sets. The segmentation may also try to provide separable clusterings for use in later node versus node priority testing. This is how the Schumacker priority list algorithm used clusters [SSS, 74].

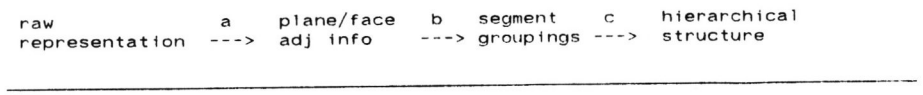
The last preprocessing transformation (c) from segmented structure into actual approximation hierarchy depends on the approximation being used. This can most often be done automatically and will be discussed later.

3.2 Hierarchy Search Algorithms

The general outline of the routines used to search through the hierarchy and aggregate the appropriate material is displayed in Fig. 2. The algorithms are written in a pseudo-Pascal with liberties taken for conceptual simplicity. The system uses the preprocessed hierarchy elements as "packages" and manipulates them as units. The expansion of these elements is done in as "lazy" a way as possible.

The top level routine constructs the hierarchy as discussed in the last section. This routine then takes a list of beams/viewing windows into the environment and applies the hidden surface removal to each one in turn. In our application, the list of beams

Fig. 1 Preprocessing Required



corresponds to a list of sound beams to be traced. The reflections generated are simply new input beams inserted into the list. In a pure graphics application, the beam list could be a succession of frames for a moving viewpoint (such as in an aviation simulator) or the succession of pixel windows for a raster display (as in [RW,80]).

Our own application includes a special node defining the receiver which is projected and clipped with the environment nodes. The detection of the receiver node in a viewing window is noted (along with other statistics) as a strike on the receiver position. We also incorporate heuristics for the beam tracing to further eliminate extraneous beams whenever possible. For example, a beam (original or image) source which has a greater distance to the receiver position than the predetermined maximum trace length can be discarded.

The routine examine is used to perform a recursive descent search into the hierarchy to produce a 'clip list' of nodes which are partially contained within the beam. The routine intersect tests the given beam for intersection against the current node. This routine depends strongly on the approximation elements used in the hierarchy and will be discussed in the next section. Examine ignores entirely excluded nodes, and continues expanding and examining partially included nodes. In order to resolve circular overlap properly the clipping routine requires separable collections of planar faces; therefore examine will not expand nodes containing only planar faces. It records these and all entirely included nodes for passing to the clipping control routine.

The clipping control routine is used to control the "laziness" of the expansion in producing a list of visible polygons. It performs a rough ordering by depth on the nodes and coalesces those which cannot have their relative depth resolved. It then calls a clipping routine based on that by Weiler and Atherton [WA,77] with the current beam and the node to be expanded and clipped. The clip routine will return the visible portions of the polygonal faces and the (possibly segmented) remainder of the beam. If the beam has been segmented, recursive calls to

Fig. 2 Hierarchy search algorithms

```
1) Top level;

  VAR
  ROOT        { root node of hierarchy,
                contains entire scene                    };

  beam_list   { list of beams / viewing windows          };

  clip_list   { list of hierarchy nodes which are
                either entirely contained in current
                beam or are planar                       };

  polygon_list
              { list of clipped visible polygons
                in the scene                             };

BEGIN
  construction(raw_input, ROOT);

  FOR all beams in beam_list DO
    BEGIN
      polygon_list <--- nil;
      clip_list <--- nil;

      examine(beam, ROOT, clip_list);
      clipping(beam, clip_list, polygon_list);
     {reflections(beam, polygon_list, beam_list)}
    END
END.
```

```
2) PROCEDURE examine(beam, node, clip_list);
   VAR status {returns the result of intersecting
               the given beam and node                   };

     BEGIN
       intersect(beam, node, status);

       CASE status OF
         entirely_excluded : {ignore};
         entirely_included : insert(node, clip_list);
         partially_included:
             IF all subnodes of node are planar
                 THEN insert(node, clip_list)
                 ELSE
                   FOR all subnodes of node DO
                       examine(beam, subnode, clip_list)
       END
     END;
```

```
3) PROCEDURE clipping(beam, clip_list, polygon_list);

     BEGIN
       depth_sort(clip_list);

       WHILE      area(beam) > epsilon
             AND  NOT empty(clip_list) DO
         BEGIN

           node_list <--- nil;
           node <---  head(clip_list);
           clip_list <--- tail(clip_list);

           IF all subnodes of node are planar THEN
              BEGIN
                clip(beam, node, polygon_list);
                IF beam is segmented THEN
                   FOR all subbeams of beam DO
                     clipping(subbeam, clip_list,
                                       polygon_list)
              END
           ELSE
             BEGIN
               FOR all subnodes in node DO
                 insert(subnode,node_list);
               clipping(beam, node_list, polygon_list)
             END
         END
     END;
```

clipping (one for each beam segment) along with the current occluding node list serve to deal with each segment as a new beam.

Note that the iteration in the clipping routine is tied to a threshold (epsilon) on the beam area. This does not cause problems in the general case; a complete solution can be formed by setting epsilon to zero. However, in the acoustics modeling, the epsilon can be used to eliminate unnecessary processing when all but an insignificant portion of the beam has been clipped. This epsilon can also come in useful in a raster scan pixel calculation where only one value is to be returned for the entire window. Processing of the window could stop at 50 or 60 per cent of the area of the original window (for example) and a weighted area value could be returned possibly providing sub-pixel resolution effects. This is related to the resolution clipping suggestion of Clark. In the event that the scene is unbounded in some directions, the clip-list will be exhausted before the beam area is and the unassigned portions of the window can be assigned some background value.

Weiler's algorithm for clipping is an extension of Sutherland's reentrant polygon clipping algorithm [SH,74]. The algorithm performs a rough depth sort on the polygonal elements and then repeatedly slices the visible section of the closest polygon out of the beam until the beam area or the list of polygonal elements is exhausted. It operates by intersecting all segments in the "clip" polygon with all the segments in the "subject" polygon and in two passes uses the intersection points and the original points to trace out the clipped polygon and the remaining beam window. If so desired, a backface test can be incorporated to eliminate even more planar candidates.

The structure of the algorithms is such that many computations can be done independently of others. This could lend itself nicely to a parallel implementation of the window processing.

4.0 Comparison of Bounding box and Convex Hull Approximations

The polyhedral elements we examine as candidates for approximation elements in the hierarchy are minimum bounding boxes and convex hulls as defined earlier. The tradeoffs between them as outlined below indicate that there is no clear-cut winner.

As a further aid to computational efficiency, we approximate the beam by its convex hull. In constructing the convex hull of the beam, the holes need not be considered. Since the k rays defining the main contour of the beam are specified in order, the hull can be constructed in O(k) time. Since all approximations considered here are convex, preliminary intersection testing to eliminate non-intersections always involves testing two convex objects, a problem which has been well studied in the computational geometry literature [CD,80] [DK,81] [MP,77] [ShH,76].

We first compare the time for construction of a hierarchy element of n points. The construction of the convex hull of a set of n points can be done in time O(n log n) [PH,77]. It appears that even if the h points on the convex hull are given, it is necessary to use time $O(h^2)$ to compute the minimum volume bounding box. This may seem initially surprising because the bounding box is a "cruder" approximation than the convex hull. The reason that the same "divide and conquer" strategy used in constructing the convex hull can't be used for the bounding box is that subresults cannot be easily combined. The bounding box of a given point set may not properly contain the bounding boxes of all subsets of that set. Note that a good approximation to the minimum volume bounding box can probably be constructed in linear time.

The space required for storage of the convex hull (as hull points and associated edges and faces) is linear in n, the number of input points. The space required for storage of a bounding box above is constant (the transformation matrix and the x, y, z offsets) regardless of the size of the input set.

Chazelle and Dobkin [CD,80] were the first to describe convex body intersection detection algorithms which operate in sublinear time. Dobkin and Kirkpatrick [DK,81] unify and extend their results by presenting a general approach to convex intersections which

produces a set of sublinear algorithms. The key to their approach is the localization of the intersection testing. A convex polygon of p points is preprocessed into a sequence of successively simpler convex approximations in O(p) time. Each polygon in the sequence strictly contains its successor. Consequently, an intersection with any member of the sequence implies an intersection with the original polygon. A sequence of this form with log p elements can be constructed giving rise to a straightforward O(log p) intersection detection algorithm.

The 3D case is almost identical. Given a convex polyhedron with p points a sequence of inner approximations is constructed. A vector or plane is tested for intersection with the sequence by performing local tests on each element in the sequence, essentially growing the polyhedron towards the testing object. There are a logarithmic number of elements and performing a local test against each (in constant time) produces the O(log p) time result. Using this hierarchical representation for two polyhedra of p and q points respectively makes possible an O(log p * log q) polyhedron against polyhedron intersection detection algorithm. Thus the convex approximation of a beam with k rays can be tested for intersection with a convex polyhedron of p points in O(log k * log p) time. Note that a standard 4-sided window can be tested against a convex polyhedron in O(log p) time.

The intersection of a vector with a bounding box can be detected in constant time by projecting the 8 points defining the box onto a backplane and testing the vector's intersection with that backplane for containment within the projected shadow of the box. This combined with the Dobkin/Kirkpatrick 2D convex intersection result can be used to determine a beam-box intersection in O(log k) time. Note that a standard 4-sided window can be tested against a bounding box in constant time.

The convex hull approximation is always contained within the bounding box approximation for a given set of points. Since the convex hull is a "tighter" fit than the bounding box, it is less likely to result in beams which intersect the approximation structure but which miss the underlying structure. The convex hull is easier to compute (exactly) than the bounding box. Convex hulls of several structures can be easily aggregated into one large containing hull. However, the bounding box approximation can be stored in constant space regardless of the complexity of the underlying structure. A bounding box also admits to a more efficient intersection algorithm. Figure 3 summarizes the convex hull/bounding box comparison of this section.

Fig. 3 Element Summary Table

	Convex Hull	Bounding Box
construction	O(n log n)	$O(h^2)$
storage	O(n)	constant
intersection with beam	O(log k * log n)	O(log k)
intersection with 4-sided window	O(log n)	constant

Notes: n - number of input points
h - number of points in convex hull
k - number of rays in hull of beam

All logarithms are base 2.

Our application is space constrained, due to the typically large nature of architectural databases. It is also time constrained due to the potentially exponential growth of beam reflections. Therefore, neither scheme has an obvious superiority.

A way to obtain some of the benefits of both is to use a hybrid hierarchy. This hybrid could use bounding boxes near the root where the space savings will be the greatest and spurious intersections, although more probable, are inexpensive. Convex hulls could be used further down in the hierarchy for their "tighter" fit approximation to decrease the amount of work passed to the clipping routines.

5.0 Summary

Section 1 introduces the motivation for the problem and discusses previous work related to geometric hierarchies. Section 2 defines our input requirements and the entities we deal with. Section 3 states the general problem in terms of these definitions. Section 3 then discusses the construction and use of hierarchical structures to solve the problem. Section 4 examines the complexities of using different geometric structures as hierarchy elements. It also presents methods of efficiently using convex approximations to improve intersection times.

Our work is continuing in applying computational geometry results to computer graphics problems [D,82] both in our application and in a general setting. We are proceeding with a Pascal implementation of the algorithms discussed in this paper. A more detailed assessment of our results will be presented elsewhere.

References

[BEH,79]
Baer, A., Eastman, C., & Henrion, M. "Geometric Modeling: A Survey" in CAD 11, No.5, 1979, pp.253-272.

[CD,80]
Chazelle, B., & Dobkin, D. "Detection is Easier Than Computation", Proc. ACM Symposium on Theory of Computing, Los Angeles, May 1980, pp.146-153.

[C,76]
Clark, J.H. "Hierarchical Geometric Models for Visible Surface Algorithms", Comm. ACM 19, No.10, 1976, pp.547-554.

[D,82]
Dadoun, N. "Hierarchical Approaches to Hidden Surface Elimination", M.Sc. Thesis, UBC, forthcoming 1982.

[DK,81]
Dobkin, D.P., & Kirkpatrick, D.G. "Fast Detection of Polyhedral Intersections", manuscript to appear.

[MP,78]
Muller, D.E., & Preparata, F.P. "Finding the Intersection of Two Convex Polyhedra", Technical Report, University of Illinois, Oct. 1977.

[PH,77]
Preparata, F.P., & Hong, S.J. "Convex Hulls of Finite Sets of Points in Two and Three Dimensions", Comm. ACM 20, No.2, 1977, pp.87-93.

[RR,78]
Reddy, D.R., & Rubin, S. "Representation of Three Dimensional Objects", Carnegie-Mellon University Technical Report CMU-CS-78-113, 1978.

[R,80]
Requicha, A.A.G. "Representations of Rigid Solids: Theory, Methods, and Systems", Computing Surveys 12, No.4, 1980, pp. 437-464.

[RW,80]
Rubin, S.M., & Whitted, T. "A 3-Dimensional Representation for Fast Rendering of Complex Scenes", Proc. SIGGRAPH 80, Computer Graphics 14, No.3, July 1980, pp. 110-116.

[Sh,78]
Shamos, M.I. "Computational Geometry", Ph.D. Thesis, Yale, May 1978.

[ShH,76]
Shamos, M.I., & Hoey, D. "Geometric Intersection Problems", Proc. 17th Annual IEEE Symposium on Foundations of Computer Science, 1976, pp.208-215.

[SuH,74]
Sutherland, I.E., & Hodgman, G.W. "Reentrant Polygon Clipping", Comm. ACM 17, No.1, 1974, pp. 32-42.

[SSS,74]
Sutherland, I.E., Sproull, R.F, & Schumacker, R.A. "A Characterization of Ten Hidden-Surface Algorithms", Computing Surveys 6, No.1, March 1974, pp. 1-55.

[WA,77]
Weiler, K., & Atherton, P. "Hidden Surface Removal Using Polygon Area Sorting", Proc. SIGGRAPH 77, Computer Graphics 11, No.2, Summer 1977, pp.214-222.

[W,79]
Walsh, J. "The Simulation of Directional Sound Sources in Rooms by Means of a Digital Computer", M.Mus. Thesis, University of Western Ontario, London, Canada, 1979.

[W,80]
Walsh, J. "The Design of Godot: A System for Room Acoustics Modeling and Simulation", paper E15.3, Proc. 10th International Congress on Acoustics, Sydney, July, 1980.

[WD,81]
Walsh, J. & Dadoun, N. "The Design and Development of Godot: A System for Room Acoustics Modeling and Simulation", presented at the 101st meeting of the Acoustical Society of America, Ottawa, May 1981.

[WD,82]
Walsh, J. And Dadoun, N. "What Are We Waiting for? The Development of Godot, II." Presented at the 103rd meeting of the Acoustical Society of America, Chicago, April 1982.