HIERARCHICAL APPROACHES TO THE HIDDEN SURFACE PROBLEM

BY

NORM DADOUN

B.Sc. (Honours) University of Western Ontario 1977

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

IN

THE FACULTY OF GRADUATE STUDIES

(Department of Computer Science)

We accept this thesis as conforming
to the required standard.

THE UNIVERSITY OF BRITISH COLUMBIA

December, 1982

© Norm Dadoun, 1982
In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Computer Science

The University of British Columbia
1956 Main Mall
Vancouver, Canada
V6T 1Y3

Date Dec 3/82
A general method for performing hidden line/surface elimination in complex domains is presented. This method uses a preprocessed hierarchical organization of the environment features. Convex polyhedra are used as elements of this hierarchy. Each polyhedron forms a convex approximation of the environment features which it contains. This hierarchy is used to provide efficient algorithms for solving the intersection, sorting and clipping subproblems associated with the hidden line/surface problem. Methods for constructing and using this hierarchy are discussed. Recent results from computational geometry are applied to element construction and intersection processing.
# Table of Contents

Abstract .................................................................................................................. ii  
Table of Contents ................................................................................................. iii  
List of Figures ........................................................................................................ v  
Acknowledgements ................................................................................................. vi  

## CHAPTER 1: Introduction ............................................................................. 1  
1.1 The Hidden Line/Surface Problem ............................................................... 2  
1.2 'Computational Acoustics' ............................................................................. 3  
1.3 Relating Acoustics and Graphics ................................................................. 7  
1.4 Thesis Outline ............................................................................................... 9  

## CHAPTER 2: Definitions ......................................................................... 11  
2.1 Environment Representations .................................................................... 11  
2.2 Beam Definition .......................................................................................... 13  
2.3 Problem Definition ...................................................................................... 15  
2.4 Approximation Structure Definitions ....................................................... 18  

## CHAPTER 3: Hierarchical Processing ...................................................... 23  
3.1 Previous Work ............................................................................................ 23  
3.2 Our Approach ............................................................................................. 25  
3.3 Processing Outline ....................................................................................... 26  
3.4 Preprocessing .............................................................................................. 27  
  3.4.1 Basic Representation ............................................................................. 28  
  3.4.2 Segmentation ......................................................................................... 33  
  3.4.3 Approximation Hierarchies ................................................................. 38  

## CHAPTER 4: Algorithms Which Use The Hierarchy ............................. 40  
4.1 Sorting ........................................................................................................ 49  
4.2 Intersection ................................................................................................. 53  
4.3 Clipping ....................................................................................................... 57  
  4.3.1 The Clipping Routine .......................................................................... 58  
  4.3.2 Previous Related Work ........................................................................ 64  
  4.3.3 Weiler's Method and Suggested Improvements ................................ 68  

## CHAPTER 5: Hierarchy Approximation Elements .................................. 75  
5.1 Two-Dimensional Approximation Elements ............................................. 76  
  5.1.1 Construction ......................................................................................... 77  
  5.1.2 Storage ................................................................................................. 78  
  5.1.3 Intersection ......................................................................................... 80  
5.2 Three-Dimensional Approximation Elements ......................................... 88  
  5.2.1 Construction ......................................................................................... 89  
  5.2.2 Storage ................................................................................................. 93  
  5.2.3 Intersection ......................................................................................... 94  
5.3 General Comparisons ............................................................................... 101  
5.4 Hybrid Hierarchies ..................................................................................... 103
CHAPTER 6: Summary, Conclusions and Further Work ........... 106

6.1 Summary .................................................. 106
6.2 Conclusions ................................................ 107
6.3 Further Work ................................................ 109
   6.3.1 Graphics .............................................. 109
   6.3.2 Geometric Modeling and Computer-Aided Design .... 111
   6.3.3 Computational Geometry .............................. 112
   6.3.4 Computational Acoustics ............................ 113

References ....................................................... 114
List of Figures

1.1 Acoustics Simulation ........................................... 6
2.1 Clipping a Polygon ............................................. 17
2.2 Minimum Bounding Polygons .................................... 19
3.1 Required Preprocessing ........................................ 28
3.2 Converting Sweep to Boundary Representations .............. 30
3.3 Segmentation Using An Articulation Point .................... 36
4.1 Proposed Algorithms - Top Level ............................ 41
4.2 Proposed Algorithms - Examine ............................... 42
4.3 Proposed Algorithms - Clipping - Clip ...................... 45
4.4 Using Intervals for Spatial Ordering ......................... 51
4.5 Sutherland-Hodgman Clipping .................................. 66
4.6 Precomputed Intersection Clipping ........................... 67
5.1 Minimum Rectangle Construction .............................. 79
5.2 Inner Sequence Representation ............................... 81
5.3 Outer Sequence Representation ............................... 83
5.4 Monotone Polygonal Chains ................................... 84
5.5 Lemma 5.2 Diagram ........................................... 86
5.6 2D Approximation Summary ..................................... 88
5.7 Bounding Box Construction Counter-Example ................ 91
5.8 Removing a Vertex ............................................. 95
5.9 Removing a Bounding Half-Space ............................. 97
5.10 3D Approximation Summary ................................... 100
Acknowledgements

Over the long haul many people have helped me out, both intellectually and spiritually; there are far too many to mention them all in this brief space.

Thanks to John Radke and the multitude of folks I've lived with for the wonder and beauty of Armadilloes and helping me to realize that I will never play hockey. To Jimbo and Iggy (and other department inhabitants) for doing sounding board imitations and helping to generate ideas. To Peg for the invaluable use of Bessie Toyota without whom I'd still be camped out in my office.

To my supervisor David K for great brainstorming sessions and the uncanny ability to focus attention on the essentials. Finally, to John Walsh for combining music, graphics, and geometry by dreaming up computational acoustics and showing me how much fun it is.
CHAPTER 1: Introduction

This thesis describes ways of structuring environment descriptions hierarchically to solve the hidden line/surface elimination problem efficiently. This thesis also brings some of the results and analytic techniques in computational geometry to bear on a computer graphics problem.

Computational geometry is a relatively new field of research which takes an algorithmic approach to classical and modern geometric problems. According to Shamos and Hoey [ShamHoey, 76, p. 208], two of the early researchers in the field, "the task of computational geometry is to relate the geometric properties of figures to the complexity of algorithms that manipulate them."

The research in this area uses the techniques of computational complexity to compare the efficiencies of different algorithms which provide a solution to a given geometric problem. By comparing the preprocessing, storage, and computation time costs of various geometric algorithms, the 'best' algorithm for a particular problem can often be identified. Although many of the results in computational geometry are ideally suited to practical application in computer graphics, most work in computer graphics has yet to take advantage of them.
1.1 The Hidden Line/Surface Problem

The hidden line/surface problem is well known in computer graphics [NewSpr,79] [FolVan,82] and arises from a desire to compute realistic two-dimensional images from three-dimensional data. In rendering a line drawing, hidden line removal refers to the identification of visible line segments and/or the exclusion of line segments occluded or hidden by opaque objects. Hidden surface removal refers to the closely related problem in doing shaded renderings on a raster or scan-line display. The first solution of either of these problems to appear in the literature was part of L. G. Roberts's [Rob,63] early work in computational vision.

A survey and categorization of existing solutions to these problems was produced in 1974 by Sutherland, Sproull, and Schumacker [SuSprSch,74]. The survey remains a relevant exegesis of approaches to these problems. The algorithms discussed are divided into three general approaches: object space algorithms, image space algorithms and list priority algorithms. The object space algorithms perform their computations to arbitrary precision without regard to display resolution limitations. The image space algorithms take advantage of known resolution limitations to limit the amount of computation associated with a rendering. List priority algorithms do not restrict their computations to object space or image space but compute relative priorities between
comparable surfaces. These priorities are used to quickly determine the visible components when doing a rendering.

We will examine the hidden surface and related problems and will present a new hierarchical organization to aid in the solution of these problems. The algorithms presented will generally operate in object space but can be modified to take advantage of image resolution limitations.

1.2 'Computational Acoustics'

A minor digression here will serve to explain the background of and motivation for our research. 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 [Wal,79] [Wal,80] [WalDad,81] [WalDad,82]. The system uses a geometric representation of a room or concert hall and provides an audible simulation of the sound field experienced at any position within this environment.

This simulation is based on the assumption that the acoustical effects within a room or concert hall can be adequately modeled by a combination of geometrical acoustics and a localized theory of diffraction [Pier,81] [Kell,58]. Geometrical acoustics models sound as emanating from a vibrating body in wavefronts. Therefore, all acoustical
effects at a given receiver position can be predicted by tracing these wavefronts as sound beams (ray bundles or pencils) from the source, through reflections, to that position.

It is essential to note the difference between a 'beam' which is a solid angle with a finite cross-section and a 'ray' which is a zero-width vector. This model is fundamentally different from other research models [KrokStrSor,68] [SchrAtal,63] in that it uses beams rather than rays. A beam when striking across an edge will 'split' into two reflected beams. A ray, being zero-width, can never 'split'.

Diffraction is the phenomenon which allows sound to 'bend' around corners. A localized edge-based model of diffraction is incorporated by noting when a sound beam strikes an edge at which diffraction can occur. That edge then becomes a secondary source position with a number of beams emanating directly from it. This model of diffraction would be difficult to incorporate in a ray-tracing system because "a ray ... would impinge upon an edge ... with probability zero" [Wal,79,p.241].

The audible simulation system is divided into three stages (see Fig 1.1):

a) The construction of a geometric model of a room using a Computer-Aided Architectural Design (CAAD) system
The current system uses a boundary representation defined by planar faces, edges and vertices. More details about our representation are given in Chapter 2; the issues of dealing with different initial representations are addressed in Chapter 3.

b) The geometric model is then used in the sound beam-tracing. A receiver position and several sound source positions are defined within the model. Sound can be thought of as emanating spherically from each source position. This expanding sphere is broken into a number of sound beams or solid angles, each with a simple polygonal cross-section. Each beam is traced through the model (along with reflection, diffraction, absorption, and related effects) to determine whether or not it will impinge upon the receiver position within a given trace length.

c) The information collected during the beam tracing is used to program a bank of digital filters. These digital filters operate on a digitized anechoic recording, processing it in the same way the room or concert hall being modeled would massage the original sound. When the processed recording is reconverted to an analog signal and replayed under controlled conditions, one hopes to provide a realistic audible simulation of what the sound would be like in the room or hall being modeled.
(a) CAAD Geometric Model of Room

(b) 'Sound Beam' Tracing

(c) Audible Simulation

Figure 1.1 Acoustics Simulation
1.3 Relating Acoustics and Graphics

The underlying geometrical problem, testing intersections related to the beam tracing, can be formulated as a variant of the hidden surface elimination problem [DadKirkWal,82] 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 most common instantiation of the traditional hidden surface problem can be considered static with one viewer position and one viewing window. Within the acoustical simulation, the problem can be considered dynamic in that it has many viewing positions and many viewing windows. Another kind of dynamic formulation of the hidden line/surface problem arises in constructing an airplane flight simulator. In this setting, a succession of views may be computed with only small changes in viewpoint. Thus, most solutions to this problem will take advantage of this 'frame coherence' to help in the computation. On the other hand, views within the simulation could successively come from widely differing viewpoints with widely differing directions.

Within the 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 intersection testing.

We propose a hierarchical organization of the room's features so that given a particular sound source (viewing) position, it is possible to eliminate quickly the irrelevant portions of the room (environment) representation. In searching for the intersection of the beam with the room, our underlying strategy has been to attempt to eliminate non-intersections as quickly as possible. More effort can then be expended examining areas within which intersections have occurred.

Our work explores the notion of a hierarchical representation in several ways. We examine the difficulty of automatically segmenting the space in various domains. We compare various structures as candidates for hierarchy elements. We apply recent results in the field of computational geometry to the construction, search and clipping processes. We propose and examine an environment representation which we will argue is useful not only for the acoustics beam tracing but for other applications as well.

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. Similarly, a room can refer to a concert hall or any environment in which the hidden surface problem is to be solved.

1.4 Thesis Outline

Chapter 2 defines a geometric model and discusses our environment representation. A beam definition is presented which is general enough to include our sound beam application and a typical graphics viewing window. The overall hidden line/surface problem is defined in terms of our environment and beam definitions. Within the problem definition, we identify and define the intersection, sorting and clipping subproblems. We define candidates for hierarchy elements: the convex hull, the rectilinear bounding box and the minimum-volume bounding box.

Chapter 3 presents previous work on hierarchical structures and discusses how our proposals incorporate and extend that work. It then identifies the steps required to preprocess a given environment into a hierarchical structure. The steps include transformations to the basic representation, segmentation, and actual construction of the approximation elements. It also discusses desirable properties of a hierarchical decomposition and suggests heuristics for
obtaining such a decomposition from different forms of input.

Chapter 4 proposes algorithms which use the hierarchies to efficiently solve the intersection, sorting and clipping subproblems identified in Chapter 2. The intersection search is performed by a recursive 'closest-first' search into the hierarchy. The sorting is performed by maintaining an interval ordering in a concatenable queue. The clipping is performed using a polygon-comparison graph representation described by Weiler [Weil,80]. The motivations and benefits of these algorithms are discussed.

Chapter 5 examines the complexities of using different geometric structures as hierarchy elements. In the spirit of computational geometry, these approximations are compared as to their construction, storage, and intersection costs. A hybrid hierarchy which can use various polyhedral structures as elements is described.

Chapter 6 summarizes our results and discusses other related applications. Suggestions for further work in various disciplines are presented.
CHAPTER 2: Definitions

This chapter presents definitions of terms and structures used in this thesis. The standard representation used in our research is first defined. We then define the hidden surface problem and identify its subproblems. Finally, the different structures proposed as candidates for hierarchy elements are defined.

This thesis will use the standard 'big O' order notation for describing computational complexity. A good introduction to computational complexity in a computer graphics setting is provided by Tilove [Til,81].

2.1 Environment Representations

A geometric model [NewSpr,79] [Reg,80] [RegVoel,82] is an abstract representation of a physical system in which the physical structure of the system is described by coordinates embedded in some metric space. We will be considering a geometric model of a static environment such as a collection of objects, a building, or an outdoor scene. Within a geometric model, information can be specified in a number of representation schemes. Requicha [Reg,80] provides an excellent survey of modeling representation schemes and related issues.
Our environment description is a boundary representation 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. An edge ring can be referred to as a polygonal contour.

A polygon which is not self-intersecting is simple. A face is a polygon which may be non-convex and may have holes. A face is described by a number of simple edge rings: a counter-clockwise edge ring specifying the outer contour and a list of clockwise edge rings specifying the contained holes. The edge directions are specified for convenience in later processing; according to these definitions the 'interior' of a face is always on the left. The face plane equation can be derived from any one of these edge rings.

If we ignore the metric space within which our environment description is embedded, then the vertices and edges define a symbolic graph relation. This is what many researchers in geometric modeling refer to as topology [EaLiSt, 75]. We can exploit this graph relation by applying some graph theoretic tools to it. The face adjacency graph is the dual graph with nodes in the adjacency graph relating to faces in the original graph, and edges corresponding to face adjacencies. An articulation point in a graph is one which when removed causes the graph to become disconnected.
Many instantiations of our representation will have a natural 'real-world' interpretation in which faces will correspond to facets of walls or solid objects. These objects can be thought of as defining a solid contained within their boundary description. In a solid-modeling scheme, this can be referred to as object coherence. This form of object coherence is not required in a boundary description of elements in a given environment. However, face orientation can be incorporated to describe the 'outside' of a wall or object. This may be used to eliminate back faces (faces oriented away from the viewer) from consideration.

2.2 Beam Definition

Within a geometric model of a scene as described above, a viewing window is defined as a viewing (source) position and an ordered list of rays leaving this viewing position delimiting a beam. This beam will have a polygonal cross-section, which may be non-convex and may have holes.

The representation of a beam is similar to that of a face; the beam is represented by a number of rays, corresponding to vertices, which are ordered in circular lists, corresponding to simple edge rings. A beam is specified by a circular counter-clockwise list of rays forming the outside contour of the beam and a list of circular clockwise ray lists defining holes within the beam contour. As in the face description, the
directions are specified for convenience in determining the 'interior' of the beam. A standard rectangular viewing screen would be modeled here as four ordered rays leaving the viewer position and defining an extended viewing pyramid.

The interior of the beam defines an unbounded solid so that questions about portions of the environment intersecting the beam are well-defined. The interior angles between any two rays in the specification of a beam will always be less than \( \pi \). Thus, it is always possible to find a plane such that the intersection of the beam and plane defines a bounded beam polygon. In most graphics applications, to avoid distortion problems in the projection, the interior angles are much less than \( \pi \).

The beam axis is defined as a semi-infinite ray extending from the viewing (source) position through an internal point of a beam polygon. For example, the centre of gravity of a beam polygon might be used to define a beam axis. The beam axis is used to define a spatial ordering on hierarchy elements as described in Section 4.1.

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 be used as input to a vector display. At the other extreme, the beam window could be regarded as a single pixel (picture element) for which a single
shading value was to be derived. Thus the extraction of values for all pixels could be used to generate a raster scan image.

One of the reasons for including non-convex beams and beams with holes is related to our acoustics application. By defining beams with this generality, we can start with a small number of large beams and let the environment do a natural decomposition. The alternative to allowing non-convex beams is to do a convex decomposition when non-convexities arise. The alternative to allowing holes is to 'split' the beams around holes if they arise. Either of these alternatives (when compounded with the beam splits resulting from the environment) could cause an unnecessary proliferation of beams. However, allowing beams to be non-convex with holes will require a more complex approach to the clipping problem defined in the next section.

Another reason for allowing beams of this generality is to facilitate deferral of processing hierarchy elements. This will be further explained in the clipping section of Chapter 4.

2.3 **Problem Definition**

In terms of the above definitions, our **problem definition** is: for a particular geometric model, identify quickly all faces or portions of faces visible within a large variety of beams with specified source positions and directions.
In our formulation of the hidden line/surface problem, three principal subproblems can be identified. The first is to identify which elements lie within the range of vision by intersecting the specified beam with the elements in the geometric model. The brute force solution to this subproblem would test all faces for containment within the viewing window. A more intelligent approach, which we explore in some detail, is to form rough clusters of faces and to test the window against each cluster.

The second subproblem is to **sort** the elements contained within the window to determine a depth ordering. This is done so that once the intersection work has determined the relevant intersections within the scene, it is possible to determine which are the 'closest' or 'visible' elements within those intersections. If two polyhedral elements do not intersect then they are called **separable**.

The third subproblem is to identify which portions of contained elements are to appear in the final rendering. This can be thought of as **clipping** visible elements against other visible elements and the window boundary to produce the final image. A **clip polygon** is a polygon to be clipped out of a given beam. A **passby beam** is a portion of a given beam which does not overlap with a given clip polygon (see Figure 2.1).

Other formulations and approaches to this problem may not
Figure 2.1 Clipping a Polygon
separate these particular subproblems. For example, an earlier approach to solving this problem in the acoustics setting combined the intersection and sorting subproblems by 'stepping' along a beam axis and testing rough clusters for intersection at each 'step'.

2.4 Approximation Structure Definitions

Our approaches make use of geometric structures used as spatial approximations of portions of the given environment. We now define some of these approximation structures.

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 an ordered circular list of vertices describing the outer contour of the hull.

A two-dimensional bounding rectangle is a rectangle which contains \( P \). This can most easily be specified by its four corner vertices. The minimum-area rectilinear bounding rectangle is the minimum-area rectangle containing \( P \) and having sides parallel to the coordinate axes. A minimum-area bounding rectangle is a minimum area rectangle of arbitrary orientation which contains \( P \) (see Figure 2.2).

The three-dimensional convex hull of a set of points in three-space is the unique minimum-volume convex polyhedron
Figure 2.2 Minimum Bounding Polygons
which contains that set of points. There are several options for representing the 3D hull. For our purposes, the simplest is a graph whose vertices are the extreme points of the convex set and whose edges denote adjacency on the convex hull. Each vertex will have associated with it an ordered ring of its adjacent vertices.

Another useful representation of the 3D convex hull is the doubly-connected edge list (DCEL) of Muller and Preperata [MulPrep,78]. In the DCEL, the 3D convex hull is represented by a list of vertices, a list of edges, and a list of faces. Each vertex has an ordered ring of incident edges associated with it. Each face has an ordered ring of incident edges associated with it. Each edge has pointers to its incident vertices and faces. The DCEL can be easily derived from the simpler representation noted earlier and, although it requires more storage, it has the advantage that information about the faces is readily available.

In our context, convex hulls are approximations of the point sets defining faces (2D) or objects (3D). In general, convex objects have nice computational properties. In recent years, a wealth of efficient algorithms for constructing and using convex hulls has been devised by researchers in computational geometry [MulPrep,78] [PrepHong,77] [Sham,78] [DobKirk,82]. For example, there is an algorithm due to Preparata and Hong to construct the 3D convex hull of n points
in $O(n \log n)$ time [PrepHong,77].

Relative to the convex hull, a **bounding box** is a much cruder approximation with a much simpler representation. Given a set $T$ of points in three-space, a bounding box is a rectangular parallelepiped which contains $T$.

The **rectilinear bounding box** is a bounding box with sides parallel to the $xy$, $xz$, and $yz$ planes. It is defined by a 'maximum' and a 'minimum' vertex where the maximum (resp. minimum) is defined by the maximum (resp. minimum) $x$, $y$, and $z$ values in $T$. The two vertices define the maximum and minimum corners of the bounding box. This structure was proposed by Eastman and Lividini [EaLi,75] for spatial search in an architectural database.

Reddy and Rubin [RedRub,78] use an **arbitrarily-oriented bounding box** as a compact spatial representation. This is a structure which retains the simplicity of the rectilinear bounding box but potentially improves the spatial approximation by freeing the bounding box from any axis dependence.

The **minimum-volume bounding box** is the next logical extension of the arbitrarily-oriented bounding box. Given a set $T$ of points in three-space, a minimum-volume bounding box is a minimum-volume arbitrarily-oriented parallelepiped which contains $T$. It can be specified as a $4 \times 4$ transformation matrix.
(into local 'box' coordinates) and x, y, and z offsets to define the limit of the box. Note that the minimum-volume bounding box of a given set of points is not necessarily unique. Above and beyond the fact that several transformation/offset combinations can describe the same box, different boxes may result in the same (minimum) volume.
CHAPTER 3: Hierarchical Processing

As part of the hidden line/surface problem definition, three subproblems (intersection, sorting and clipping) were identified. We propose using a new geometric hierarchical organization of elements within a given environment to provide good algorithms for solving each of these subproblems.

This chapter will describe previous work done on geometric hierarchies. It will then describe the motivation and structure of our hierarchy. Finally, the preprocessing necessary to convert basic representations to hierarchical organizations will be discussed.

3.1 Previous Work

The use of geometric hierarchies in structuring domain information for hidden surface algorithms was first suggested by Clark [Cl,76]. He proposed using hierarchies to limit the amount of information under consideration at any particular time (borrowing the notion of a 'working set' from operating system theory) and also discussed generalizing the idea of clipping to include resolution clipping, motion clipping and other optical effects. He suggested that search algorithms take the form of a recursive descent through the hierarchy.

Eastman and Lividini [EaLi,75] used rectilinear bounding
boxes to do crude (minimax) spatial search testing in an architectural database. In their earliest work, they did not describe a hierarchical organization but used one subsequently in the BDS (Building Description System) [EaLiSt,75]. BDS was a research architectural design system which concentrated on representational and database issues. The hierarchical organization was carried over into the design of GLIDE (Graphical Language for Interactive DEsign) [EaHEn,77]. This is a Computer-Aided Architectural Design system which is, in part, the basis of the hierarchical representation originally used in the acoustics modeling system [Wal,79].

Independently, Reddy and Rubin [RedRub,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 [RubWhit,80]. They used a hierarchy of arbitrarily-oriented bounding boxes 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.
3.2 Our Approach

In our domain, a given geometric model will be interrogated many times. Although many different beams may be under consideration, the searching and intersection problems will be essentially the same. To reduce the cost of each search and thereby reduce the overall time, our desire is to trade search time for preprocessing time. Our approach is to use a hierarchical organization of approximating structure as an alternate representation of the scene. Although this hierarchical organization may take different forms, for brevity in the following discussion, it will be referred to as the hierarchy.

The hierarchy is composed of polyhedral elements, each of which contains a portion of the environment. Each polyhedral element is a convex approximation of the portion of the environment which it contains. The hierarchy can be viewed as a description of the environment at various levels of detail. Some questions about the original scene can be answered quickly by using only a very coarse description.

In a typical configuration, 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. A leaf may take one of several forms. A leaf may be a
'degenerate' three-dimensional (i.e. two-dimensional) convex approximation of one of the planar face elements making up the scene. It is also possible that a leaf may be a procedural definition of a portion of the environment as described in section 3.4.1.

Our work extends that of Rubin and Whitted in several ways. The hierarchy we describe can use various convex approximation elements and can also incorporate procedural definition of portions of the environment. Rubin and Whitted use approximation hierarchies in a hidden surface raster scan setting only. This raster scan setting uses the hierarchy to derive pixel values only and therefore does not deal with clipping polygonal elements. To demonstrate the use of a hierarchy in a hidden line setting, we present a way of clipping that exploits the hierarchical representation.

3.3 Processing Outline

The processing involved in solving the hidden-surface 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 intersection, sorting, 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 section; items 5-6 are examined thereafter.

3.4 Preprocessing

What form might the input data take? On one hand, Rubin and Whitted assume a "point or surface representation ... [not] aggregated into any hierarchical form" [RubWhit, 80, p. 113]. At the other end of the spectrum, Walsh describes: "a hierarchy of walls (and/or wall elements), planar faces, [edges,] and vertices" [Wal, 79, p. 236]. Walsh's representation was intended to be the output of a CAAD system and thus would already have 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 the initial structure of the data but most users will require some 'massaging' of data into appropriate forms.
Figure 3.1 Required Preprocessing

All of this could be done manually using structure editors as Rubin and Whitted 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 examine ways to deal with each. Figure 3.1 identifies the appropriate portions and labels the necessary transformations.

3.4.1 Basic Representation

With the multitude of representations in general use [Req,80] the first transformation (a) would map the input into a geometric vertex boundary surface representation. Boundary representation is the most popular representation scheme in computer graphics although many other schemes exist and are used in different applications. Some of these schemes are pure
primitive instancing, spatial occupancy enumeration, Constructive Solid Geometry (CSG), and (translational or rotational) sweeping. A full discussion of representation schemes is beyond the scope of this thesis but as Requicha suggests "multiple representations [are necessary] for achieving a degree of versatility and efficiency higher than that of extant modelers" [Req,80,p.46]. Efficient exact algorithms do exist for performing conversions between some representations. For example, there are exact algorithms for converting from a sweep to a boundary representation. For other conversions, such as CSG to boundary, only approximation algorithms are known.

The basic transformation may involve some sort of conversion from 2-dimensional input data to a 3-dimensional representation. For example, the test rooms dealt with in the acoustics modeling [WalDad,82] are 'built' using two-dimensional digitizer input to a sweep program. The sweep or extrusion representation is then converted to the required boundary representation (see Figure 3.2).

For dense gridded data, the method of Fowler and Little [FowLit,79] can be used to construct a Triangulated Irregular Network (TIN). A TIN is a digital terrain model which is made up of non-overlapping triangular facets of varying size. Given a number of points defined on a grid, their algorithm will construct automatically a TIN which uniformly fits those points
Figure 3.2 Converting *Sweep* to Boundary Representations
to any specified tolerance.

The associated object structure from some representations (for example, the primitive object descriptions in a Constructive Solid Geometry representation) can be used to guide the hierarchical segmentation later on. In some cases, a non-planar boundary surface may have to be approximated by planar faces.

Once the boundary representation has been constructed, the face adjacency dual graph can be derived automatically. This, for many domains, will be useful in the segmentation portion of the preprocessing.

Many design systems [BaEaHen,79] have a parameterized object definition facility. It is possible to define the details of an object only once by tying some features of the object to parameters to be specified at instantiation time. Then various instantiations could be placed in the environment by specifying position, orientation and specific values for the parameters.

For example, much storage space (and tedium) could be saved in the design of a concert hall by specifying the design of a seat once and then placing instantiations at various places in the hall. In this way, not only is there a saving in time and space but the parameterized objects provide a level of
abstraction which would aid the design process. In the above example, it could allow the designer to easily rearrange the seating plan without regard to the unimportant details of the geometric consequences of such an action.

A parameterized object definition is one example of a procedural representation. There is another form of non-explicit representation which can also be useful in our application of acoustics simulation: a portion of the environment which is known to reflect sound in a particular way can be treated as a 'black box' with no explicit representation of the details of its contents. Whenever a beam enters its immediate vicinity from a given range of angles, 'reflected' beams with directions and properties determined by some table look-up could be generated [Wal,79].

This 'black box' approach could be used to incorporate portions of the environment which are rendered procedurally according to some stochastic process model [FourPuCarp,82]. If the outer bounds of this 'black box' are specified then the inner detail can be specified by a pointer to a set of procedures. These procedures when invoked would compute the required surface details.

In the types of procedural representation schemes described above, it would be unwise, and (in some cases) impossible to insist on preprocessing the procedurally defined
portions of the environment into an explicit face, edge, vertex form. If the outer bounds of the procedurally defined portion of the environment were specified, then the interior could be treated as an indivisible segment within the hierarchy. This 'special' segment could be treated as being entirely specified by its outer boundary for purposes of 'wrapping' in an approximation element. The segment would then only be 'unwrapped' or, in the procedural setting, invoked if the bounding element were found to lie within some viewing window.

3.4.2 Segmentation

Transformation (b) involves the recursive partitioning of the faces and edges in the scene into segment groupings. The segmentation/aggregation transformation takes the overall environment description as specified by a boundary representation and produces segmented clusters which actually define the hierarchy elements. Each segment grouping will then be wrapped within an approximation element as discussed in the next section.

Obviously, a cityscape or concert hall description has a different fundamental description from a digital terrain model of a mountainous scene and its segmentation will require different heuristics. For example, the cityscape description may be expected to have a regularity in the angles between faces which can be exploited. It is unreasonable to expect an
algorithm that will provide a good segmentation for any given scene regardless of domain. We can examine tools to use and properties to exploit for various domains. The list of heuristics presented is not meant to be exhaustive but is meant to convey the kinds of approaches one might take to automatically segment an environment description.

There are a number of desired properties which this segmentation should have to be useful for the hidden surface processing. Unfortunately not all of these will be possible for all scenes. In intersecting the viewing window with the hierarchy elements, we wish to eliminate as much of the environment as quickly as possible. Therefore, the segmentation should have a fairly low branching factor from any level to the children of that level.

The segmentation should attempt to balance some measure of size (complexity, volume, or aspect ratio) among elements at the same level of the hierarchy. The segmentation might be driven by a reduction in the overall volume or the sizes of point sets. As well, it is desirable to have separability of individual elements so that an ordering can be obtained unambiguously and in some cases that a relative ordering can be precomputed for use in later node versus node priority testing. This was how the Schumacker priority list algorithm used clusters [SuSprSch,74].
For many scenes, segmentation alone will not be sufficient. At various points, aggregation will be necessary: sometimes a large number of disjoint elements will be input or will result from a single partition. Thus, a simple top/down or bottom/up approach will not work; a combination of the two will be necessary. For example, the first partition of a room may result in many disjoint elements. To avoid a high branching factor in the hierarchy, some of these objects will have to be associated together to provide an intermediate hierarchy level.

The connected components of the adjacency (dual) graph can be used as an initial partition of the environment. The resulting segments can then be further decomposed using various heuristics depending on the composition of the scene. 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 (see Figure 3.3).

In a similar way, high degree vertices in the dual can be used as 'cluster points' for segmenting. This might correspond to a large surface with many adjacencies, such as a wall with many adjacent objects. This is useful even if the resulting branching factor is still high because the neighbouring objects can be regrouped together by aggregating disjoint elements.
(a) Sample Scene

(b) Face Adjacency Graph of (a) with Articulation Point

Figure 3.3 Segmentation Using An Articulation Point
For aggregation another form of adjacency graph can be used to include faces (or more complex elements) that are 'close' to each other, using spatial proximity to define the graph relation.

To segment the scene, Rubin and Whitted suggested that standard statistical feature detection techniques be used to locate automatically the complex portions of the scene. An example of this is to search for peaks in a three-dimensional histogram of vertices or edges within the environment.

Another segmentation technique is to use a plane as a bounding half-space to partition a group of elements. This plane could be formed by a least-squares fit on the environment vertices. The orientation of the partition plane could be determined by a Hough-transform type voting scheme [DudaHart,72]. This voting scheme could use environment face orientations weighted with surface areas. In this scheme, buckets would be set up corresponding to various orientations. The list of faces would be scanned and a face falling into an appropriate orientation range would have its face area added to the corresponding bucket. Once all the faces had been scanned, the maximum would define the desired orientation.

When aggregating elements near to the planar face level, some elements could be associated on the basis of orientation as to their visibility from a number of different viewpoints.
This could be thought of as illumination or shadowing from a number of viewpoints.

3.4.3 Approximation Hierarchies

The last preprocessing transformation (c) from segmented structure into actual approximation hierarchy depends on the approximation being used. Once the segmentation has been completed, each of the segment clusters will be replaced with an approximation element. These approximation elements can be considered as packages surrounding a portion of the environment. The portion of the environment being packaged can be defined explicitly, or procedurally.

Note that in the case of a parameterized definition of objects within the environment it is sometimes possible to parameterize the approximation element along with the object definition. Thus the time and space saving inherent in the parameterized definition can be inherited by the hierarchy building. For example, if a seat within a concert hall design is defined as a parameterized object then the approximation element can be 'built' around the parameterized seat. An instantiation of the seat would also correspond to an instantiation of the approximation element.

The desirable properties for approximation elements are efficient construction, compact storage, efficient algorithms
for manipulation, and a minimum of wasted volume (which will be referred to as 'tightness of fit'). Given the portion of the environment to be packaged, the construction of the geometric approximation being used can most often be done automatically, as will be discussed in Chapter 5.
CHAPTER 4: Algorithms Which Use The Hierarchy

This chapter proposes a set of algorithms to solve the hidden line/surface problem. These algorithms use the hierarchy described in Chapter 3 to deal with the previously identified intersection, sorting and clipping subproblems. The algorithms use the preprocessed hierarchy elements as 'packages' and manipulate them as units. The basic idea, and the source of the algorithms' efficiency, is that expansion of these elements is done in as 'lazy' a way as possible.

The general outline of the control routines used to search through the hierarchy and aggregate the appropriate material are shown in Figures 4.1 to 4.3. The algorithms are written in a pseudo-Pascal with liberties taken for conceptual simplicity. The routines presented assume the existence of standard list-processing routines such as empty, head, tail, list, append and others. Any routines which are not self-explanatory will be described in the text.

The reader is cautioned that although the algorithms are presented in an ersatz programming language, they have not been implemented. Furthermore, the algorithms are proposals for ways of using the described hierarchical structure which we envision as being suitable for our application. It is quite possible, and entirely likely, that different applications may require different approaches to one or more of the identified
PROGRAM Top_level:

VAR
  ROOT ' { root node of hierarchy, contains entire scene };
  element_list { list of hierarchy nodes under consideration, ordered by closest point to current viewing position };
  interval_list { list of defined intervals for each hierarchy node in element_list, used to order nodes in element_list };
  beam_list { list of beams / viewing windows };
  beam { current beam / viewing window under consideration. };
  beam_axis { axis of the current beam. };
  beam_proj_plane { plane orthogonal to current beam axis, used for projecting environment elements. };
  windowed_polygon_list { list of clipped visible polygons in the scene };
  passby_list { list of beams defining 'background' portion of scene, ie. unimpeded portion of current beam. };

BEGIN
  construction(raw_input, ROOT);
  initialize(beam_list);
  WHILE NOT empty(beam_list) DO
    BEGIN
      beam <- head(beam_list);
      beam_list <- tail(beam_list);
      get_beam_axis(beam, beam_axis, beam_proj_plane);
      element_list <- list(ROOT);
      interval_list <- list(interval(ROOT));
      windowed_polygon_list <- NIL;
      passby_list <- NIL;
      examine(beam, element_list, interval_list, windowed_polygon_list, passby_list);
      render(beam, windowed_polygon_list, passby_list);
      (reflections(beam, windowed_polygon_list, beam_list))
    END
  END.

Figure 4.1 Proposed Algorithms - Top Level
PROCEDURE examine(beam, element_list, interval_list, windowed_polygon_list, passby_list);

VAR

clip_list { list of planar elements to be clipped out of beam / viewing window. };

clip_int_list { ordering on list of planar elements used by clipping routine derived from interval_list sort. };

mid_element { element found by scan_separable to be the first element after the first separable sequence of elements. Used in SPLIT to partition interval seq. };

beam_passby_list { list of beams resulting from testing current beam against first (closest) hierarchy node. };

ret_passby_list { returned passby list from recursive call to examine: the passbys on this list have been tested against everything and are truly background. };

passbeam { the beam from passby_list currently under consideration. };

status { contains the result of intersecting the current beam and node };

front_list { portion of the environment which is closest to the viewing position and which can be determined separable by a simple interval test. };

front_int_list { interval list describing interval ordering on elements in front_list. };

Figure 4.2 (a) Proposed Algorithms - Examine
BEGIN

{ If all the elements in the element_list are planar then there are no elements to be expanded. The element list is split into separable portions and the closest portion is clipped out of the input beam. The passby portions of the beam are then recursively applied to the remainder of the element list. }

IF all elements in element_list are planar THEN
BEGIN

    scan_separable(element_list, interval_list, mid_element):;

    SPLIT(mid_element, element_list, clip_list, element_list);
    SPLIT(interval(mid_element), interval_list, clip_int_list, interval_list);

    passby_list <--- NIL;

    clipping(beam, clip_list, clip_int_list, windowed_polygon_list, beam_passby_list);

    IF NOT empty(element_list) AND NOT empty(beam_passby_list) THEN
        FOR all passbeams in beam_passby_list DO
            BEGIN
                examine(passbeam, element_list, interval_list, windowed_polygon_list, ret_passby_list);
                append(passby_list, ret_passby_list)
            END
    ELSE IF NOT empty(beam_passby_list) THEN
        passby_list <--- beam_passby_list
END

Figure 4.2 (b) Proposed Algorithms - Examine (cont.)
ELSE

{ If there exist non-planar elements in the element list then choose
the first non-planar element and DELETE it from the element list.
Then expand that node testing each of its subnodes for intersection
with the current beam. If the subnode intersects the beam at all
and is not a planar back face then reinsert it into the element list. }

BEGIN
  expand_node ← first_non_planar(element_list);
  expand_node_interval ← interval(expand_node);
  DELETE(expand_node, element_list);
  DELETE(expand_node_interval, interval_list);
  FOR all subnodes of expand_node DO
    BEGIN
      intersect(beam, subnode, status);
      CASE status OF
        entirely_excluded : {ignore};
        entirely_included, partially_included:
          IF NOT(planar(subnode) AND backface(subnode)) THEN
            BEGIN
              INSERT(subnode, element_list);
              INSERT(interval(subnode), interval_list)
            END
        END
      END
    END;
  END;

{ Use the interval list to SPLIT the element list into 2 separable
sublists and recursively examine the closest. If there are passby
beams and untested portions of the environment then recursively
test them against each other. }

  scan_separable(element_list, interval_list, mid_element);
  SPLIT(mid_element, element_list, front_list, element_list);
  SPLIT(interval(mid_element), interval_list, front_int_list, interval_list);
  passby_list ← NIL;
  examine(beam, front_list, front_int_list, windowed_polygon_list, beam_passby_list);
  IF NOT empty(element_list) AND NOT empty(beam_passby_list) THEN
    FOR all passbeams in beam_passby_list DO
      BEGIN
        examine(passbeam, element_list, interval_list, windowed_polygon_list, ret_passby_list);
        append(passby_list, ret_passby_list)
      END
    END
  ELSE IF NOT empty(beam_passby_list) THEN
    passby_list ← beam_passby_list
  END

END:

Figure 4.2 (c) Proposed Algorithms - Examine (conc.)
PROCEDURE clipping(beam, clip_list, clip_int_list, windowed_polygon_list, beam_passby_list);
VAR
  clip_poly \{ Polygon which has been chosen as the polygon to clip out of the beam window by this call to clipping. Beam is segmented into portion inside clip_poly and portion outside clip_poly. \};
  clip_polygon_in_beam \{ List of windowed polygons resulting from clipping clip_poly against beam. ie. portions of clip_poly visible within beam. \};
  beam_not_in_clip_poly \{ List of passby beams resulting from clipping clip_poly out of beam. ie. portions of beam not striking clip_poly. \};
  status \{ contains the result of intersecting the beam and the clip_poly. \};
  passbeam \{ The beam from beam_not_in_clip_poly currently under consideration for recursive call to clipping. \};
BEGIN
  \{ Choose clip_poly to be the closest visible polygon and use intersect to make sure that a portion of it lies within the beam area. If it does then call clip to slice clip_poly out of beam. Repeat until there is an area clipped out of the beam or there are no more polygons to clip. \}
  IF area(beam) > epsilon THEN
    BEGIN
      REPEAT
        clip_poly \----- NIL;
        clip_polygon_in_beam \----- NIL;
        beam_not_in_clip_poly \----- NIL;
        IF NOT empty(clip_list) THEN
          BEGIN
            clip_poly \----- head(clip_list);
            DELETE(clip_poly, clip_list);
            DELETE(interval(clip_poly), clip_int_list);
            intersect(beam, clip_poly, status);
            IF status <> entirely_excluded THEN
              clip(beam, clip_list, clip_int_list, clip_poly, clip_polygon_in_beam, beam_not_in_clip_poly)
          END
        UNTIL (clip_poly = NIL) OR NOT empty(clip_polygon_in_beam);
    END
  Figure 4.3 (a) Proposed Algorithms - Clipping - Clip
If we have exhausted the clip list and no clip_poly exists which intersects the input beam then the whole beam is returned as the passby. If such a clip_poly does exist then clip_poly_in_beam is inserted into the windowed polygon list and the polygons in beam_not_in_clip_poly define the passby list. We can use recursive calls to clipping to clip the passby beams against the remaining polygons.

```plaintext
IF (clip_poly = NIL) THEN beam_passby_list <--- beam
ELSE
BEGIN
  beam_passby_list <--- NIL;
  append(windowed_polygon_list, clip_poly_in_beam);
  IF NOT empty(beam_not_in_clip_poly) THEN
    FOR all passbeam in beam_not_in_clip_poly DO
      clipping(passbeam, clip_list, clip_int_list,
               windowed_polygon_list, tmp_passby_list):
      append(beam_passby_list, tmp_passby_list)
  END
END
END;
```

PROCEDURE clip(beam, clip_list, clip_int_list, clip_poly,
               clip_poly_in_beam, beam_not_in_clip_poly);

VAR

test_list  { List of polygons within beam with intervals which overlap test_poly. }

poly  { Polygon under consideration for insertion into test_list as being inside beam and having overlapping intervals with clip_poly. }

intersection_points  { The intersection points of all the polygons in the test list along with the beam. Used to build the graph representation used for the actual clipping. }

graph  { Graph structure of polygon contours and crossings. Defines tessellation of the plane into regions with list of polygon owners. }

search_owner  { Constraint passed to traverse_graph defining attributes of regions which are returned. }

Figure 4.3 (b) Proposed Algorithms - Clipping - Clip (cont.)
contour_list
{ List of contours returned from traverse graph.
The owner list of each contour with satisfy the
search_owner constraint. }

contour
{ Polygon from contour_list currently under
consideration. }

BEGIN

{ Build up the test_list of possibly occluding polygons from the polygons
in clip_list with overlapping intervals. }

    test_list = list(clip_poly);
    FOR all poly in clip_list DO
        IF overlap(interval(clip_poly), interval(poly)) THEN
            append(test_list, poly);

{ Build up Weiler's graph representation for doing generalized clipping.
Traverse the graph to determine the areas of clip_poly which lie inside
the beam (i.e., contours which have clip_poly and beam as owners). Test each
of these areas against its other owners for a depth conflict and discard
the ones for which clip_poly doesn't win. Then traverse the graph once
more for the portion of the beam outside clip_poly. }

    intersect_polygons(beam, test_list, intersection_points);
    build_graph_representation(beam, test_list, intersection_points, graph);
    search_owner_list = owner_list(clip_poly AND beam
        AND any(test_list));
    traverse_graph(graph, search_owner_list, contour_list);

    FOR all contour in contour_list DO
        IF occluded(beam, clip_poly, contour) THEN
            remove_owner(graph, contour, clip_poly);
        search_owner_list = owner_list(clip_poly AND beam
            AND any(test_list));
        traverse_graph(graph, search_owner_list, clip_poly_in_beam);

    IF NOT empty(clip_poly_in_beam) THEN
        BEGIN
            search_owner_list = owner_list(beam AND NOT clip_poly);
            traverse_graph(graph, search_owner_list, beam_not_in_clip_poly);
        END

END;

Figure 4.3 (c) Proposed Algorithms - Clipping - Clip (conc.)
Within the proposed algorithms, the top level routine constructs the hierarchy as discussed in the last section. This routine then takes a list of windows into the environment and identifies the visible surfaces within each in turn. In the acoustics application, the list of beams corresponds to a list of sound beams to be traced. The reflections generated are simply new input beams appended to the list. In a pure graphics application, the beam list could be the 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 [RubWhit, 80]).

In the acoustics beam-tracing, a special node defining the receiver is projected and clipped with the environment nodes. The detection of the receiver node in a viewing window is noted as a strike on the receiver position. The total path length through reflections to a hit on the receiver, the reflecting materials' effects on the frequency response of the path, and the direction of strike on the receiver position are other pieces of relevant information. We also incorporate heuristics for the beam tracing to further eliminate extraneous beams whenever possible. For example, consider the predicted distance from a beam's source, through reflections, to the receiver position. If this distance is greater than the predetermined maximum trace length then that beam can be
discarded.

The control routine \textit{examine} is used to perform a recursive descent search into the hierarchy to extract and aggregate collections of planar faces which are partially or completely contained within the current beam. It uses a routine called \textit{intersect} to test a particular beam for intersection with any given hierarchy node. The sorting is performed within the \textit{examine} routine by maintaining a sorted list of hierarchy element intervals. After using these intervals to identify two sets of planar faces which are separable, a routine called \textit{clipping} clips the identified faces out of the current beam in the order of their appearance.

4.1 Sorting

The sorting incorporated into this kind of hierarchical processing should be as simple as possible and should attempt to aid in the 'laziness' of the hierarchical package processing. To this end, the sorting is used to identify some of the separable portions of the hierarchy which have been expanded and discovered to lie within the current beam.

The sorting uses the beam axis and a projection plane orthogonal to this axis to define an ordering on the environment elements. To maintain the beam axis, and therefore the ordering, during segmentation of the beam, the beam axis is
calculated only once for each top level call of examine.

Each hierarchy element occupies an interval along the beam axis defined by the element's minimum and maximum distance from the source (viewing) position. This minimum and maximum can be the minimum and maximum points on the associated approximation element rather than the true minimum and maximum. As will be shown in Chapter 5, these can be derived from a preprocessed 3-dimensional convex hull in time logarithmic in the number of hull points and from the respective bounding boxes in constant time.

If the endpoints of these intervals are sorted then the non-overlapping collections of intervals can be identified by a single sweep through the sorted endpoints. This is done by maintaining a count while sweeping across the sorted endpoints (see Figure 4.4). Starting with zero, add one whenever encountering an 'opening' point of an interval and subtract one whenever encountering a 'closing' point of an interval. Whenever the count becomes zero, an interval corresponding to a separable portion of the environment has been identified. This use of intervals is essentially the overlapping segments problem described by Shamos [Sham, 78, p. 123]. In Shamos's formulation, given m intervals, the overlapping segments can be discovered by sorting the endpoints in O(m log m) time and then performing an O(m) sweep across the sorted endpoints.
(a) Defining Distances along Beam Axis

(b) Corresponding Intervals

Figure 4.4 Using Intervals for Spatial Ordering
These intervals will be used to order hierarchy elements for intersection and clipping. The clipping must be able to isolate subsequences of intervals corresponding to separable collections of faces in the environment. It must also be able to access the sorted elements in order. When a hierarchy element is expanded into its contained sub-elements, the new intervals corresponding to those elements must be inserted into the interval list. When an element is found to lie outside the beam under consideration, its interval must be deleted from the interval list. The ordering on the intervals must be maintained dynamically through these insertions and deletions.

A concatenable queue can be implemented using a balanced tree scheme [AhHopUll,74] (such as a 2-3 tree) to meet these criteria. This data structure supports INSERT, DELETE, FIND, CONCATENATE and SPLIT instructions while maintaining its leaves in order. A sequence of k of these instructions can be processed in $O(k \log k)$ time. Thus, m intervals can be sorted in $O(m \log m)$ time using m INSERT instructions and the separable intervals can still be identified in $O(m)$ time. In the following discussion, we assume that the intervals will be maintained in such a structure. The capitalized INSERT, DELETE and SPLIT instruction within the algorithm outline refer to operations on a structure of this form.

The use of intervals is an approximation which attempts to reduce a spatial sorting problem to a one-dimensional sorting
problem. This, in essence, treats the beam as a planar wavefront which is orthogonal to the beam axis. A spherical wavefront, while being a more realistic model, introduces computational difficulties. The farthest polyhedron point from a given viewing position cannot be determined using a unimodal search [AvTouBha, 81]. However, the maximal polyhedron point in a particular direction can be obtained using a unimodal search.

The ordering is obtained by noting at what distances from the source position any hierarchy element enters and leaves the wavefront (see Figure 4.4). This simplification does not recognize pairs of elements which are separable but which happen to have an overlapping interval. This could be dealt with by doing more extensive preprocessing to precompute separating planes between neighbouring segments as in the method due to Schumacker et al described in [SuSprSch, 74]. These separating planes could be used in addition to the interval ordering. However, simply getting a good hierarchical decomposition would tend to encourage separability. Another factor related to the separability of hierarchy elements, which we will return to in the next chapter, is the 'tightness of fit' of the hierarchical approximation element used.

4.2 Intersection

Examine is the main control routine which decides which elements are to be excluded summarily and which are to be
retained and expanded for further testing (see Figure 4.2). Examine takes as input the specification of a beam, a portion of the environment described in an element list and the interval ordering for these elements. It returns as output a list of clipped polygons corresponding to the planar surfaces visible within the beam. It also returns the 'passby' list: a list of beams defined by the portions of the beam not intersecting any of the portion of the environment passed to examine in the element list.

It is possible that the viewing position will be contained within one or more hierarchy elements. These elements must be expanded before any others so that the portions 'behind' the viewing position can be discarded and the portions 'in front' can be inserted into the interval list. Here, we can assume that some routine invoked before the first call to examine handles this special case and that the viewing position is not internal to any of the elements in the element list.

At any particular time in the search, elements of widely varying sizes and complexity may be stacked for examination. The recursion is invoked in such a way that the search strategy becomes a closest-element depth-first search into the hierarchy. The use of separable intervals and 'passby' beams is what allows the evaluation of distant elements to be deferred.
The routine always splits the input portion of the environment into a 'closer' portion and a 'farther' portion. It will always finish dealing with the 'closer' portion before it examines the 'farther'. In this way, if the 'closer' portion entirely occludes the beam, no time is wasted in processing the 'farther'. Examine maintains and uses the sorted interval order described in the last section to decide where to split the environment. The interval order is also used to decide which non-planar elements to expand.

If examine is called with a sequence of elements, all of which are planar, then no recursion need be used to expand any of them. Once these elements are clipped out of the beam, the routine terminates. Examine uses the intervals to isolate the first separable collection of planar faces, and SPLITs the element list into two separable subsequences. A call to the clipping routine, with the closer subsequence, will then segment the beam into the portions striking the planar faces in that subsequence and those passing by.

The portion of the beam which is not occluded by the given sequence of faces is returned from the clipping routine as a (possibly empty) list of 'passby' beams. If such a list of passby beams is ever returned from clipping or from a recursive call to examine then each passby beam is dealt with in a separate recursive call to examine with the remaining sequence of elements. Passby beams which remain after all sequences
have been dealt with are returned to the calling routine.

In the event that the scene is unbounded in some directions, the element sequence will be exhausted before the passby list is. In this case, the passby list will be returned to the main calling routine which can assign the portions of the window corresponding to the passby beam list some background value.

If, in the initial test in \texttt{examine}, not all the nodes correspond to planar elements then the routine takes the first nonplanar element and expands it into a number of subnodes. \texttt{Examine} uses the routine \texttt{intersect} to test the current beam for intersection against each given subnode. The intersection algorithm used in this routine depends on the approximation structure used in the hierarchy and will be discussed in the next chapter.

\texttt{Examine} ignores subnodes which lie entirely outside the beam. It \texttt{INSERT}s all other subnodes into the element list ordered on their closest-point proximity to the source position. It also \texttt{INSERT}s the included subnode's interval into the interval list. \texttt{Examine} uses a backface test to eliminate planar elements oriented away from the viewing position. It then uses the interval list to determine the first separable interval. In the worst case, this interval list may correspond to the entire element list. The element list is then \texttt{SPLIT}
into two separable portions and each portion is recursively 'examined'.

4.3 Clipping

One of the benefits of the hierarchical structure presented is the ability to defer processing on potentially complex portions of the room. The interval ordering on the hierarchy elements allows us to unwrap the closest separable portion of the environment and deal entirely with it before continuing on to elements farther away. For example, one large polygonal face may occupy all of the beam area. Processing on the potentially complex portions of the scene behind it should be avoided until this face is clipped out of the current beam.

The clipping approach used is critical to the successful use of the hierarchy for deferring processing. It must be able to deal with collections of polygons which are not necessarily separable. It must be able to defer processing the portions of the window which do not contain any of the polygons currently under consideration but which may contain elements stacked by examine.

The hierarchical approach limits the number of polygons under consideration for clipping so that most of the calls to this routine should involve a very small number of clip-list polygons. However, it is possible that in some scenes a large
number of polygons will remain and the method used for clipping should be general enough to handle this as well.

The method proposed for clipping meets these criteria. Weiler's method for polygon comparison [Weil, 80] takes as input a number of polygons to be 'compared' and builds a graph representation of the polygon contours and their intersections. By traversing this graph representation, it is possible to derive a list of polygonal contours along with their 'owning' polygons. This method and some suggested improvements will be described further in Section 4.3.3.

4.3.1 The Clipping Routine

This section describes generally the outline of the routines used for clipping. The clipping method is described in more detail in subsequent sections.

The proposed clipping routines are presented in Figure 4.3. The clipping routine takes as input a beam defined by a source (viewing) position along with delimiting rays emanating from that position. It also requires a 'clip-list' sequence of polygons along with an interval list as described in the sorting section above to specify their relative ordering. All clipping performed by this routine takes place on a projection plane chosen to be orthogonal to the beam axis. The beam's intersection with this plane defines a 'beam polygon' against
which the other polygons are clipped.

All of the defined polygons are simple but any can be non-convex with holes. As stated in the definition in Chapter 2, all polygon edges are directed so that the interior of the polygon is on the left. This means that the polygon outer boundaries are specified in counter-clockwise order and that the polygon hole boundaries are specified in clockwise order. This also provides a way of specifying the 'visible' side of a clipping boundary.

The **clipping** routine returns a list of 'visible' polygons clipped from the original clip-list polygons. It also returns a list of new passby beams defined by the portion of the original beam which did not intersect any of the clip-list polygons. Each of the polygons in the output polygon list must correspond to the closest surface to the viewing position for its particular projection area on the projection plane. The output defines a decomposition of the beam polygon in that the projection area of the visible polygons plus the area of the passby beam polygons equals the area of the original beam polygon.

Note that the entry into the **clipping** routine can be 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 processed.

This epsilon may also be 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 some fraction of the area of the original window (for example) and a weighted area value could be returned possibly providing sub-pixel resolution effects. As Clark [Cl,76] suggested, this uses the hierarchy scheme to limit processing according to the resolution limitations of the display. Clark referred to this as resolution clipping.

In expanding the hierarchy elements in examine, a rough depth sort, as described in Section 4.1, is maintained in the form of intervals. Because of the way the polygons were aggregated in examine for the call to clipping, the list of clip polygons does not have a separable subsequence of intervals. This does not mean that the clip-list polygons do not admit to a total depth ordering but only that the interval approximation could not determine an ordering relative to this viewing position.

Even though there are no separable subsequences in the interval list, the intervals still provide useful information. For example, note that a given polygon can only have a depth
conflict with other polygons which have overlapping intervals. Thus in processing the clip-list, it is sufficient to sweep from front to back considering polygons one at a time. A polygon is only tested for a depth conflict against the polygons following it in the ordering which have an overlapping interval.

The initial loop in clipping tries to obtain a polygon which is visible within the beam. Subsequently, the clip routine will be used to slice the visible portions of this polygon out of the current beam. The first polygon in the ordered interval list is considered as a candidate for a visible polygon. It is tested for intersection with the beam. If there is no intersection then this polygon is discarded and the routine loops back to test another.

If there is an intersection with the beam then clip is called to partition the beam into the visible portion of the clip polygon and the passby portion of the beam. Even if the polygon lies within the beam, it is possible that the entire polygon is occluded. If no portion of the polygon under consideration is visible then it is discarded and the routine loops back to test another polygon for visibility within the beam.

After exiting from this loop, if no portions of any of the input polygons are visible within the beam, then the entire
beam is returned as a passby. Otherwise, some portions of one of the polygons is visible within the beam and those portions are appended to the windowed polygon list. If there are portions of the beam outside the partially visible polygon (defining passby beams) then they are handled individually with recursive calls to clipping.

Clip takes as input the current beam, the input polygon list, the corresponding interval list and the current polygon polygon to be clipped (referred to in the following discussion as the clip polygon). It produces a list of visible polygons sliced from the clip polygon and a list of polygons which define the passby portions of the beam which lie outside the visible portions of the clip polygon.

The clip routine uses a graph representation due to Weiler [Weil,80] to partition the clip polygon into visible and non-visible portions. It is also used to partition the beam into the portions which intersect and the portions which passby the clip polygon. This graph representation defines a tesselation of the projection plane into all possible regions defined on the projections of the beam and the polygons to be clipped.

Each region has associated with it a list of owning polygons. Initially, any polygon which contains a particular region is said to be an owning polygon of that region. By
'traversing' the graph representation, it is possible to determine a list of regions satisfying any 'owning polygon' requirements. For example, the list of regions which are owned by the beam and the polygon to be clipped defines the portions of the polygon which are potentially visible within the beam. Since the graph representation includes all possible regions defined on the polygons under consideration, within each region there is a total depth ordering of the polygons which own that region. Thus, the single polygon which is visible within a given region of the graph representation can be determined using the plane equations of that region's owning polygons and any ray from the viewing position passing through that region. Weiler's graph representation, its construction, and maintenance will be further discussed in Section 4.3.3.

The clip routine first aggregates a list of test polygons which have an overlapping interval with the clip polygon. These polygons are the only polygons in the clip list which could possibly occlude any portion of the clip polygon. Weiler's graph representation is constructed on this list of test polygons, the clip polygon, and the beam polygon.

The first traversal of the graph representation produces the complete list of regions which lie inside the beam and the clip polygon (and possibly others). The next loop tests each region to see if the clip polygon is the 'visible' (closest) polygon containing that region. If the clip polygon is not the
visible polygon within that region then it is removed as an owner of that region.

The graph is traversed again to obtain the regions defined by the visible portions of the clip polygon. This second traversal is performed to avoid unnecessary segmentation of the visible clip polygon area. It produces the minimum number of regions defining the visible portions of the clip polygon within the beam.

The final traversal obtains the regions which are within the beam but not the clip polygon. This again produces the minimum number of regions for which this is true. These regions will be used to define the passby list for this beam.

4.3.2 Previous Related Work

The method used for clipping is based on an algorithm by Weiler [Weil,80]. This was an abstraction of the method presented by Weiler and Atherton [WeilAth,77]. Their algorithm for clipping is, in turn, an extension of Sutherland and Hodgman's reentrant polygon clipping algorithm [SuHod,74]. It is instructive to examine these algorithms briefly before returning to Weiler's method in detail.

Sutherland and Hodgman's algorithm [SuHod,74] operates only on windows defined by convex polygons. It treats each
window edge as a bounding half-plane and clips the viewed polygons against one half plane at a time. Its input is a ring of vertices defining a polygon and a line defining a bounding half plane with a 'visible' side specified. Its output is one or more polygon vertex rings.

The algorithm operates by noting that the endpoints of a directed polygon edge can relate to a bounding half-plane in only four ways. It considers each edge in turn and takes appropriate actions for each case (see Figure 4.5). These are:

a) both endpoints visible: output first endpoint.

b) first endpoint visible and second endpoint not visible: output visible endpoint and intersection of segment with clipping boundary.

c) first endpoint not visible and second endpoint visible: output intersection of segment with clipping boundary.

d) neither endpoint visible: output nothing.

As Sutherland and Hodgman discuss, minor changes to the routine will allow the correct processing of polygons which may result in degenerate cases. This includes the clipping of concave polygons, polygons with holes, or self-intersecting polygons against a bounding half-plane. The viewing window however is always restricted to being convex.

Note that the points at which transitions occur are the
intersection points of the bounding half-plane defining the clipping boundary and the polygon to be clipped. If these points are precomputed then they can be used to trace out the visible portion of the polygon (see Figure 4.5).

Weiler and Atherton's clipping algorithm [WeilAth,77] is an extension of this precomputed intersection polygon clipping algorithm. Note that in tracing the clipped object, the clipping boundary may as well have been a polygonal chain as a straight line. The next observation, which motivates their algorithm, is that the clipping boundary can be a closed polygonal chain. In fact, the clipping boundary may be as complex as the polygon to be clipped. Thus, this clipping
Note that the direction on the clipping boundary is used to trace visible portion of the clip polygon; the interior is on the 'left'.

Figure 4.6 Precomputed Intersection Clipping
algorithm is general enough to clip non-convex polygons with holes against other non-convex polygons with holes.

Their clipping algorithm operates by intersecting all segments in the polygon to be clipped with all the segments in the polygon defining the clip boundary. In two passes, it uses the intersection points and the original points to trace out the clipped polygon and the remaining portions of the window. They note that additional processing is necessary to determine strictly containing relationships between polygons and/or holes but do not give details of the method which they used.

4.3.3 Weiler's Method and Suggested Improvements

Weiler [Weil,80] abstracted the clipping method of Weiler and Atherton into generalized 'polygon comparison' by noting that the clipping of several polygons could be 'batched' by constructing a graph representation of their vertices, edges and intersection points. This graph representation can be traversed to derive the union, intersection or difference of any combination of the input polygons.

This graph representation links edges at their intersection points as the Weiler/Atherton method did. This can be thought of as 'overlaying' all of the polygons to define a number of regions in the plane. Each region is contained within some number of the original input polygons. These
original polygons are called the 'owners' of the region. Every edge bounds two regions and has an associated list of 'owners' for each. Thus, starting at an intersection point and following the edges with the appropriate 'owners', it is possible to trace out any of the desired polygon combinations.

This method has the advantage that edge-edge, edge-endpoint, endpoint-endpoint, and coincident edge intersections are all inserted into the graph representation easily and then handled uniformly. Additional processing is required to handle non-intersecting regions arising from holes or strictly contained polygons. Weiler describes a scheme by which contour relationships can be maintained although he does not specify the containment test to be used.

To construct the graph representation, it is necessary to compute the intersections of the polygons which define the regions. In intersecting the polygons, we can continue with the general philosophy of eliminating non-intersections if possible. By quickly detecting that a pair of polygon approximations do not intersect, we can avoid executing a more complex polygon intersection algorithm with the corresponding polygons. The next chapter will discuss the method and complexity of detecting non-intersections between the convex approximations of a pair of polygons.

Up to this point, all intersection testing, sorting and
other processing has been performed on an approximation of the beam and approximations of the elements (including polygonal faces) in the hierarchy. These approximations will be described in the next chapter. Having used these approximations to quickly test for non-intersections, we can now afford to spend more time examining in detail the elements which correspond to the approximations which have passed our tests.

Shamos [Sham,78] [ShamHoey,76] presents an algorithm for computing line-segment intersections. This algorithm has some unusual properties. It may not find all intersecting pairs of line segments but it is guaranteed to find the left-most. Thus, it is suitable as a non-intersection elimination algorithm.

The algorithm proceeds as follows: Pick a direction line (say, the x direction) and sort the projections of the endpoints onto this line (the x coordinates). Sweep from left to right maintaining a list (ordered on y-coordinates) of 'current segments'. Do this by inserting a segment into the list whenever its opening endpoint is encountered and deleting it from the list whenever its closing endpoint is encountered. Whenever an insertion or deletion operation is performed, test the segments which have become adjacent in the ordering for intersection.
For \( n \) line segments, the endpoints can be sorted in \( O(n \log n) \) time. The sweep has a constant number of operations per endpoint for an \( O(n) \) time complexity. Thus, the overall asymptotic complexity is \( O(n \log n) \).

In a practical sense, the sorting step of the above intersection algorithm is not wasted because it can be incorporated into the actual intersection testing. For \( n \) line segments, there may be as many as \( n^2 \) intersections so in the worst case we cannot do better than this. However, the sorted order can be incorporated in an algorithm which finds all intersections. An intersection algorithm using the sorted order would operate in better expected time than simply intersecting all pairs of segments.

In his doctoral thesis [Sham,78,p.140], Shamos poses the question: "Of \( N \) line segments, suppose that \( k \) pairs intersect. Does there exist an \( O(\max(N \log N, k)) \) algorithm to list them?" To the best of our knowledge, this is still an open question.

Several improvements to the proposed clipping algorithms of Figure 4.3 could be made by maintaining Weiler's graph representation through recursive calls to the clipping routine. This was not incorporated in the outline for the sake of clarity. One benefit would be that in the processing of different clip polygons the graph structure would not need to be entirely rebuilt each time. The graph structure would have
new polygons inserted and clipped polygons deleted.

Another benefit is that if the graph structure is to be maintained then it is possible to do some preprocessing to facilitate the testing to be performed on it. In particular, it is possible to triangulate the defined regions by inserting edges with 'null' owners into the graph representation. This triangulation can be performed using the method of Garey et al [GarJoPrepTar,78]. If there are \( q \) points defining the regions, the regions can be triangulated in \( O(q \log q) \) time.

Having been triangulated, the regions can be further processed to accommodate the planar point location algorithm of Kirkpatrick [Kirk,81]. This algorithm with \( O(q) \) preprocessing time and \( O(q) \) increase in space, can locate the containing region of a query point in \( O(\log q) \) time.

This is useful for several reasons. The containment test of non-intersecting polygons can be performed quickly. As well, new polygons can be inserted fairly quickly. This can be done by inserting one edge at a time. The endpoints can be placed in the appropriate triangles in the triangulation and the edge's intersections can be discovered by 'walking' along the edge through the triangulation. As each 'true' edge is crossed, the intersection is updated. When all new edges have been inserted, only 'contaminated' regions need be retriangulated.
Kirkpatrick's method preprocesses the triangulation by creating a 'triangulation hierarchy'. This is a sequence of triangulations where the initial triangulation is the original, and each element after this is derived from its predecessor by removing a fixed-fraction of low-degree independent vertices and retriangulating. This continues until the final element in the sequence has a constant number of regions. Kirkpatrick shows that if there are initially q points defining the regions, there will be log(q) elements in this sequence.

The location of a point's containing region is performed by first testing the query point for location in the last triangulation in the sequence. Once the point has been located in one of the constant number of regions, its containing region is refined using the finer preceding triangulation in the sequence. This continues until the point is located within one of the triangles in the original triangulation. A constant number of tests for each of a logarithmic number of elements produces the O(log q) search result.

Problems may be introduced when trying to maintain this 'triangulation hierarchy' dynamically. As edges are inserted, the 'contaminated' regions will be retriangulated and the hierarchy elements containing these regions must be updated. This may be done quickly if the changes are localized in the triangulation but extensive alterations may require rebuilding the triangulation hierarchy entirely.
The hierarchical refinement technique used in this algorithm will reappear in the convex approximation intersection algorithms of the next chapter.
CHAPTER 5: Hierarchy Approximation Elements

In this chapter, candidates for hierarchy approximation elements are examined. The two-dimensional elements are rectilinear bounding rectangles, minimum-area bounding rectangles, and two-dimensional convex hulls. The three-dimensional candidates are rectilinear bounding boxes, minimum-volume bounding boxes, and convex hulls. These structures have been chosen as the best convex candidates for hierarchy elements. Their definitions and a discussion of their relevant features appear in Chapter 2. Here, the elements are compared as to construction time, storage cost and intersection time.

In the first section, the two-dimensional elements are discussed. In the second section, the three-dimensional candidates for hierarchy elements are compared. In the next section, we present a general comparison of the various elements. We conclude the chapter by arguing for a hybrid hierarchy where the approximation element used is chosen according to its 'tightness of fit'.

In solving the intersection subproblem defined in Chapter 2, it is necessary to compute which polyhedral approximation elements intersect a specified beam. In solving the clipping subproblem defined in Chapter 2, it is necessary to determine which polygonal approximations lie within a specified
beam.

Although the beam defines a three-dimensional volume, it is basically an extrusion of a two-dimensional object. All beam cross-sections are congruent, therefore the beam can be considered a two-dimensional object sweeping through space. Since the intersection of a beam with any projection plane orthogonal to the beam direction and on the correct side of the viewing position will always define the same polygon, it makes sense to refer to this as the beam polygon.

As mentioned in the introduction, our underlying philosophy is to eliminate non-intersections as quickly as possible. Therefore, to eliminate an approximation element from consideration, it is sufficient to detect a non-intersection between the beam and that element.

5.1 Two-Dimensional Approximation Elements

As explained above, the beam cross-section can be considered a two-dimensional structure. Therefore, the beam and face polygons are the two-dimensional entities to be wrapped in a convex approximation.

As noted earlier, the two-dimensional objects can be considered degenerate cases of the similarly defined three-dimensional structures. We will consider them separately
here because some results, notably the linear convex hull and minimum-area bounding rectangle construction algorithms, benefit from being embedded in the plane and do not extend to three dimensions.

Although a polygon in our scheme may have holes, only the outer contour is used to construct an approximation element. This is because the elements are all convex. In the following discussion, n refers to the number of vertices defining a polygon.

5.1.1 Construction

Shamos [Sham, 78, p. 45] proves an $\Omega(n \log n)$ lower bound on the two-dimensional convex hull problem by demonstrating that any solution to this problem can be used to sort. However, it is possible to do better if the input points are defined as being ordered on a simple polygonal contour. Graham and Yao [GrYao, 81] present an $O(n)$ algorithm to find the hull of an ordered simple polygonal contour.

The rectilinear bounding box can be constructed by performing a single pass across all the points and recording the maximum and minimum x, y and z values. Thus, it is produced in $O(n)$ time.

Freeman and Shapira [FrSh, 75] prove Lemma 5.1 which leads
to Theorem 5.1 (after Boyce et al [BoyDobDryGui,82]).

Lemma 5.1 [FrSh,75,P.411]:

The rectangle of minimum area enclosing a convex polygon has a side collinear with one of the edges of the polygon.

Theorem 5.1:

The minimum-area bounding rectangle of a convex polygon can be constructed in $O(n)$ time.

Proof:

The minimum-area rectangle can be found by constructing and testing the $O(n)$ possibilities satisfying Lemma 5.1. This can be in $O(n)$ time by using the following algorithm:

a) Choose an edge on the polygon and construct the minimal-area bounding rectangle which includes that edge using some projective technique such as Sloan [Sl,80]. With respect to this edge, let $a$, $b$, and $c$ denote the leftmost, farthest opposite, and rightmost vertices respectively (see Figure 5.1). Record the three vertices $a$, $b$, and $c$ defining the other rectangle sides.

b) Construct the next rectangle on the next clockwise edge by 'sliding' each of $a$, $b$, and $c$ in a clockwise direction until each has found its maximum in its particular direction with respect to this base edge.

c) Repeat b) for all edges recording the minimum-area rectangle until all edges have been examined. Q.E.D.

5.1.2 Storage

The space required for storing the two-dimensional convex hull may be linear in $n$. In contrast, the bounding rectangles can each be stored in constant space by storing the four corner points defining them. These would be stored as vertices for the polygonal face approximations
Figure 5.1 Minimum Rectangle Construction
5.1.3 Intersection

First note that the intersections of two bounding rectangles can be calculated in constant time. Next, consider the problem of doing convex hull intersections. Chazelle and Dobkin [ChazDob,80] were the first to describe convex body intersection detection algorithms which operate in sublinear time. Dobkin and Kirkpatrick [DobKirk,82] unify and extend their results by presenting a general approach to convex intersections in two and three-dimensions. The key to both approaches is the localization of the intersection testing and the preprocessing of the convex bodies which facilitates this localization.

A convex polygon of n points can be preprocessed into a sequence of successively simpler convex approximations in O(n) time. Dobkin and Kirkpatrick [DobKirk,82] describe two different sequence representations with similar properties: the inner sequence and the outer sequence. The original polygon is the first element of both sequences. Each subsequent element is a simpler version of its predecessor. The operations used to form elements in the two sequences are dual; an element in the inner sequence is formed by removing vertices while an
Figure 5.2 Inner Sequence Representation

The inner sequence representation of a polygon \( P \) is a sequence \( P_1, \ldots, P_k \) such that the original polygon \( P \) is the first element \( P_1 \) of the inner sequence. Each element \( P_i \) is a smaller simplified version of its predecessor \( P_{i-1} \), formed by removing alternate vertices (see Figure 5.2). The last element of the sequence \( P_k \) has a constant number of vertices. Each polygon in the inner sequence strictly...
contains its successor. Therefore an intersection with any member of the sequence implies an intersection with the original polygon. Since a fixed fraction (in fact, nearly half) of the vertices are removed from any element to its successor, there are $O(\log(n))$ elements in the sequence.

The outer sequence representation of a polygon $P$ is a sequence $P_1, \ldots, P_k$ such that the original polygon is the first element $P_1$ of the outer sequence. Each element $P_j$ is a larger simplified version of its predecessor $P_{j-1}$ formed by removing alternate bounding half-planes (see Figure 5.3). The last element of the sequence $P_k$ has a constant number of vertices and bounding half-planes. Each polygon in the outer sequence is strictly contained by its successor. Therefore a non-intersection with any member of the sequence implies an non-intersection with the original polygon. Again, there are $O(\log(n))$ elements in the sequence.

Dobkin and Kirkpatrick prove the following Theorem from which the corollary follows immediately.

Theorem 5.2 [DobKirk,82]:

Given the inner or outer sequence corresponding to a polygon $P$ of $n$ vertices and a line $L$, we can compute in $O(\log n)$ operations the intersection of $L$ and $P$. 
Figure 5.3 Outer Sequence Representation
Figure 5.4 Monotone Polygonal Chains
Corollary:

The intersection of a two-dimensional convex hull of n vertices with a bounding rectangle can be computed in $O(\log n)$ operations.

Dobkin and Kirkpatrick [DobKirk,82] present a "shadowing" technique which can be used to break the hull-hull intersection problem into two pairs of polygonal chain intersections (see Figure 5.4). A monotone polygonal chain (MPC) is a sequence of vertices and edges given in order of increasing y-coordinate. The bottom and top edges are extended into semi-infinite rays. A convex polygon can be described as a right MPC and a left MPC. The intersection of two convex hulls can be demonstrated by showing the intersection of the right MPC of one with the left MPC of the other and vice-versa. Note that the use of MPCs can also be used to detect a strictly containing intersection. This is done by noting where on the polygonal contours the intersection occurs.

Lemma 5.2:

Consider 2 MPCs, $P_i$ and $Q_i$, the ith elements of 2 outer sequences. Say they intersect at a point $d$. If the edge on $P_i$ containing $d$ is replaced by the corresponding edge and its neighbours on $P_{i+1}$, then $Q_i$ intersects $P_{i+1}$ iff it intersects these edges on $P_{i+1}$.

Proof:

Call the new edges in $P_{i+1}$, $x_1$, $x_2$, and $x_3$ (See Figure 5.5). The endpoints of $x_1$ and $x_3$ not on $x_2$ are points in both $P_i$ and $P_{i+1}$. Call them $s$ and $t$. The edge replacement we are concerned with in going from $P_i$ to $P_{i+1}$ is really replacing one series of edges from $s$ to $t$ with
Figure 5.5 Lemma 5.2 Diagram

another series of edges from s to t. We distinguish 2 cases:

i) s and t are on opposite sides of $Q_i$. Then $Q_i$ must intersect the new sequence between s and t.

ii) s and t are on the same side of $Q_i$. Then there are either 2 or 0 intersections. If there are 0 intersections then by the convexity of $Q_i$, $P_i$, and $Q_i$ do not intersect (as in Figure 5.3). Q.E.D.

Theorem 5.3:

Given 2 convex polygons $P$ and $Q$ of p and q vertices respectively represented by their outer sequences, we can determine a point in their intersection or detect a non-intersection in at worst $O(\log(p)\log(q))$ operations.

Proof:

As mentioned earlier, the intersection problem can be recast as a pair of MPC intersection problems. We
consider PL and QR. The two convex polygonal chains must intersect twice or not at all. In the following, we refer to PL as P and QR as Q.

The algorithm described maintains the upper intersection while alternately stepping through the P sequence and the Q sequence. Call the intersection of \( P_i \) and Q the point g. We shall proceed by first finding the upper intersection of \( P_{i_1} \) and \( Q_i \) then similarly finding the intersection of \( P_{i_1} \) and \( Q_{i_1} \).

Choose the segment of P containing g. Replace this segment with the corresponding constant number of segments in \( P_{i_1} \). By Lemma 5.2, if Q intersects P then \( Q_{i_1} \) must intersect one of these segments. Use the algorithm of Theorem 5.2 to determine the intersection of these segments with \( Q_{i_1} \). Call this intersection h. This h is then the intersection point of \( P_{i_1} \) and \( Q_{i_1} \).

Similarly, use the segment containing h in Q to find the intersection between \( Q_{i_1} \) and \( P_{i_1} \).

The initial intersection between \( P_{i_1} \) and \( Q_{i_1} \) can be found in constant time. Each 'step' through the sequence requires \( O(\log(p) + \log(q)) \) operations. There will be \( \min(\log(p), \log(q)) \) steps to be taken for \( O(\log(p) * \log(q)) \) operations overall.

Q.E.D.

It is worth noting that Chazelle and Dobkin [ChazDob,80] present a convex polygon intersection detection algorithm which operates in at worst \( O(\log(p+q)) \) time. Their algorithm uses localized testing and will terminate early if an intersection is detected. The outer sequence intersection testing will terminate early if a non-intersection is detected. If a non-intersection is more probable than an intersection, then the outer sequence algorithm is preferable. Thus, the \( O(\log(p) * \log(q)) \) non-intersection detection result of Theorem 5.3 is of practical and theoretical interest.
Convex Hull | Bounding Rectangle
---|---
construction | $O(n)$ | $O(n)$
storage | $O(n)$ | constant
intersection with convex hull beam | $O(\log m \times \log n)$ | $O(\log m)$
intersection with rect. window | $O(\log n)$ | constant

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

All logarithms are base 2.

Figure 5.6 2D Approximation Summary

A table summarizing the quantifiable results of this section appears in Figure 5.6.

5.2 Three-Dimensional Approximation Elements

The entities to be wrapped in an approximation element are three-dimensional collections of planar faces. Note that in constructing these approximation elements, if the faces have
been approximated by their two-dimensional convex hulls then only those hull points need be considered. Although this does not affect the asymptotic complexity, it may lead to a simpler implementation.

In the following discussion, \( n \) refers to the number of vertices of the hierarchy element to be approximated and \( h \) refers to the number of hull points on a three-dimensional convex hull under consideration.

5.2.1 Construction

We first compare the times for construction of a polyhedral hierarchy element of \( n \) points. The construction of the three-dimensional convex hull of a set of \( n \) points can be done in time \( O(n \log n) \) [PrepHong,77]. This algorithm exploits the fact that separable convex hulls can be combined in linear time. It uses a divide and conquer technique to build the convex hull by recursively halving the point set, finding the hull of each half and combining the results in linear time. Note that in constructing a hierarchy of convex hulls, the cost of building the hulls can be shared if the constituent elements are linearly separable.

The rectilinear bounding box can of course be constructed in \( O(n) \) time by simply examining each vertex and recording the maximum and minimum \( x, y, \) and \( z \) values.
It seems reasonable to expect a three-dimensional analogue to the Freeman and Shapira result [FrSh, 75] mentioned in the discussion of the minimum-area bounding rectangle construction in the last section. This conjecture might be that the minimum-volume bounding box of a set of points has at least one face which is coplanar with the three-dimensional convex hull defined on that point set. This might lead to an algorithm similar to the one presented in Theorem 5.1.

Unfortunately, a counter-example can be easily constructed (see Figure 5.7). Define a planar rectangle (ABCD). Above and below the rectangle, place two edges (EF and GH respectively) which lie in two planes which are parallel to ABCD. Place them such that their projections onto the ABCD plane are orthogonal, are both contained within ABCD, and intersect. Connect the edges to the rectangle forming a solid (as in Figure 5.5). Finally, note that it is possible to choose the area of ABCD to be so large as to force the distance between EF and GH to be one of the dimensions of the minimum-volume bounding box. Note that in such a box EF and GH would lie on opposite faces and that no faces of this convex polyhedron are coplanar with the bounding box.

In a sense, the fact that EF and GH lie in parallel planes suggest that taken together they define a 'supporting plane' analogous to the 'coplanar face' conjecture. This supporting plane then could be defined by two opposite edges or a face and
Figure 5.7 Bounding Box Construction Counter-Example

an opposite vertex. Avis et al [AvTouBha, 81] prove that the sequence of distances defined on a convex polygon is not unimodal. Using this result, it is easy to show that it is not possible to choose an edge and search for its farthest opposite edge using a unimodal search technique.

Thus, any constructive approach for the minimum-volume bounding box analogous to the minimum-area bounding rectangle
result would have to consider pairs of edges. Even if the \( h \) points and \( O(h) \) edges on the convex hull are available, each edge might have to be paired with a fixed fraction of the total number of edges of the polyhedron producing \( O(h^2) \) candidates to test. Each candidate could be constructed in linear time producing an \( O(h^3) \) construction result to determine the minimum-volume bounding box. No better constructive method is readily available.

This result 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 cannot be used for the bounding box is that subresults cannot be easily combined. The minimum-volume bounding box of a given point set may not properly contain the minimum-volume bounding boxes of all subsets of that set.

Note that an approximation to the minimum-volume bounding box can be constructed reasonably efficiently. This can be done by using a hill-climbing optimization routine [Win,77] to find the transformation which minimizes the volume of the rectilinear bounding box. The only drawback to this scheme is that the optimization routine may converge on a local minimum rather than the global minimum desired.

Another reasonable approximation to the minimum-volume
bounding box could be constructed using principal components or principal axes analysis [BallBro,82]. Given a set of points in space, this can be used to find a set of orthogonal basis vectors which would then define an orientation for the bounding box. These basis vectors define the orthogonal directions of maximum variance rather than the directions of absolute maximum variation. This means that given the orientation defined by the principal components, we would still have to do a linear sweep through the points to define the actual approximation to the minimum volume bounding box.

The complexity of these bounding box approximation schemes cannot really be compared with those of the other construction algorithms. These algorithms use fundamentally different approaches than the ones outlined earlier and no immediate insight is to be gained from their comparison.

5.2.2 Storage

The space required for storage of a minimum-volume bounding box is constant (the transformation matrix and the x, y, z offsets) regardless of the size of the input set. The space required for storage of the rectilinear bounding box is also constant, the minimum and maximum x, y, and z values. The space required for storage of the convex hull (as hull points and associated edges and faces) may be linear in n, the number of input points.
5.2.3 Intersection

As discussed in the introduction to this chapter, a beam is really a two-dimensional entity. Therefore to do beam/approximation element intersection testing it is sufficient to test for two-dimensional intersections on the appropriately chosen projection plane as previously described.

The inner and outer sequences of Dobkin and Kirkpatrick [DobKirk,82], as described in the last section, can also be defined on polyhedra. In the following discussion, an independent set of vertices in a polyhedron is one in which no two vertices are adjacent. Similarly, an independent set of bounding half-spaces in a polyhedron is one in which no two half-spaces intersect to form an edge of the polyhedron.

The inner sequence representation of a polyhedron \( P \) is a sequence \( P_1, \ldots, P_k \) such that the original polyhedron \( P \) is the first element \( P_1 \) of the sequence. An element in the inner sequence \( P_i \) is a smaller simplified version of its predecessor \( P_{i-1} \), formed by removing a number of low-degree independent vertices. At each step a fixed fraction of the total number of vertices is removed. The last element \( P_k \) is a polyhedron with a constant number of vertices and faces. As in the two-dimensional inner sequence, each element strictly contains its successor. Thus, an intersection with any element of the sequence guarantees an intersection with the original
(a) \( v \) is vertex to be removed

(b) After removal, new faces are formed

Figure 5.8 Removing a Vertex
polyhedron. Note that in removing a low-degree independent vertex it may be necessary to form a constant number of new faces defined on its adjacent vertices (see Figure 5.8).

The outer sequence representation of a polyhedron $P$ is a sequence $P_1, \ldots, P_K$ such that the original polyhedron $P$ is the first element $P_1$ of the sequence. An element in the outer sequence $P_i$ is a simplified larger version of its predecessor $P_{i-1}$, formed by removing a number of low-degree independent bounding half-spaces. At each step, a fixed fraction of the total number of bounding half-spaces are removed. The last element of the sequence is a polyhedron with a constant number of faces and vertices. As in the two-dimensional case, each element is strictly contained by its successor. Thus a non-intersection with any element of the sequence guarantees that there will be no intersection with the original polyhedron.

Each bounding half-space is a constraint defining one polyhedron face. Removing a bounding half-space in constructing the outer sequence is relaxing a constraint. The adjacent half-spaces intersect to form a constant number of new vertices (See Figure 5.9). In actually constructing an outer sequence, care must be taken to ensure that the removal of a bounding half-space does not produce an unbounded result. It can be shown that this can always be done such that the simplest element of the sequence will have at most six sides.
w is resulting vertex when bounding half-space is removed.

Figure 5.9 Removing a Bounding Half-Space

To detect beam/element non-intersections, we consider the outer sequence representation of the convex hull. The outer sequence of a convex polyhedron implicitly represents the outer polygonal sequences of all projections of that polyhedron. Thus, to perform a non-intersection detection using the outer sequence of a polyhedron, we can use a variant of the two-dimensional algorithm described in Theorem 5.3 of the last section.
The variation arises in that replacing a removed bounding half-plane of a polygon is not exactly the same as replacing a bounding half-space of a polyhedron and reforming the projection (see Figure 5.9). There are new vertices formed when a bounding half-space is removed. This means that in moving from one element of the sequence to its predecessor, a constant length sequence of projection vertices is replaced by another constant length sequence. The next lemma which parallels Lemma 5.2 deals with this variation.

**Lemma 5.3:**

Given a projection plane $A$ and a projection direction $d$, consider 2 MPCs, $Q_i^*$, the $i$th element of a polygonal outer sequence, and $P_i$, the projection of the $i$th element of a polyhedral outer sequence onto $A$ in the direction $d$. Say they intersect at a point $d$. If the edge sequence on $P_i$ containing $d$ is replaced by the corresponding edge sequence on $P_i^*$, then $Q_i^*$ intersects $P_i^*$ iff it intersects these edges on $P_i^*$.

**Proof:**

Note that the endpoints of the two edge sequences are the same in $P_i$ and $P_i^*$. If these are referred to as $s$ and $t$, the same argument as in Lemma 5.2 can be applied. Q.E.D.

**Theorem 5.4:**

Given a projection plane $A$ and a projection direction $d$, consider a line $L$ lying in $A$ and a convex polyhedron $Q$ of $q$ points represented by its outer sequence. In $O(\log(q))$ operations it is possible to find an intersection point or detect a non-intersection between $L$ and projection of $Q$ onto $A$ in the direction $d$.

**Proof:**

Making the appropriate substitutions in Theorem 5.2
Corollary:

The intersection of a bounding rectangle and the projection of a three-dimensional convex polyhedron of \( n \) vertices can be computed in \( O(\log(n)) \) operations.

Theorem 5.5:

Given a convex polygon \( P \) of \( p \) points and a convex polyhedron \( Q \) of \( q \) points along with their outer sequences, then in \( O(\log(p) \times \log(q)) \) operations it is possible to find an intersection point or detect a non-intersection between \( P \) and the coplanar projection of \( Q \).

Proof:

Making the appropriate substitutions in Theorem 5.3 for the result in Lemma 5.3, the same argument can be applied. Q.E.D.

The intersection of the convex hull approximation of a beam with a bounding box can be detected by projecting the 8 points defining the box onto a backplane and using the result of Theorem 5.2. If the convex hull has \( m \) vertices then this can be done in \( O(\log(m)) \) operations. Note that the intersection of a bounding rectangle and the projection of a bounding box can be performed in constant time.

To conclude this section, we present a result due to Dobkin and Kirkpatrick [DobKirk,82] referenced in the interval sorting section of Chapter 4:
100

Convex Hull  Rectilinear Bounding Box  Minimum Volume Bounding Box

<table>
<thead>
<tr>
<th>Construction</th>
<th>Rectilinear Bounding Box</th>
<th>Minimum Volume Bounding Box</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(n \log n)$</td>
<td>$O(n)$</td>
<td>$O(h)$</td>
</tr>
<tr>
<td>$O(n)$</td>
<td>constant</td>
<td>constant</td>
</tr>
<tr>
<td>$O(\log m \cdot \log n)$</td>
<td>$O(\log m)$</td>
<td>$O(\log m)$</td>
</tr>
<tr>
<td>$O(\log n)$</td>
<td>constant</td>
<td>constant</td>
</tr>
</tbody>
</table>

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

All logarithms are base 2.

Figure 5.10 3D Approximation Summary

**Lemma 5.4:**

Let $p_i(d)$ be the maximal vertex of $P_i$ in the direction $d$ where $P_i$ is the $i$th member of an inner or outer hierarchy for a polygon or polyhedron. Then, either $p_{i-1}(d) = p_i(d)$ or $p_{i-1}(d)$ is one of the new neighbours of $p_i(d)$ in $P_{i-1}$.

**Theorem 5.6:**

The maximum vertex in a given direction $d$ of a polygon or polyhedron $P$ with $p$ points can be derived from the inner or outer representation in $O(\log p)$ operations.

Given a three-dimensional convex hull of $h$ points, the maximum point in a given direction can be derived from an inner or outer sequence in $O(\log(h))$ operations. The maximum
bounding box point in a given direction can be determined in constant time by testing all (8) possibilities.

The table in Figure 5.10 summarizes the quantifiable results of this section. The final section discusses several of these results.

5.3 General Comparisons

We can now discuss some general comparisons as to the relative merits of the various approximation candidates. Our discussion will concentrate on the three-dimensional approximations: convex hull, rectilinear bounding box and minimum-volume bounding box.

The notion of 'tightness of fit' defines an ordering on approximation elements based on their respective volumes on a given point set. Let CH denote the convex hull, MVBB denote the minimum-volume bounding box, and RBB denote the rectilinear bounding box. Given a set X of points in three-space:

\[ \text{volume(CH}(X)) \leq \text{volume(MVBB}(X)) \leq \text{volume(RBB}(X)). \]

The rectilinear bounding box will, in general, form the worst fit of X because of its arbitrary dependence on axis orientation. The minimum-volume bounding box is an improvement because of its axis independence but may still define much wasted space. The convex hull is the most compact
approximation because it is (by definition) the best convex fit possible. Since the convex hull is a 'tighter' fit than either of the bounding boxes, a beam intersecting the convex hull is more likely to intersect the underlying structure than a beam intersecting one of the bounding boxes.

Tightness of fit is an important consideration which affects not only intersections but also the ability to determine an absolute relative ordering of components quickly. The expansion routine described in Chapter 4 makes use of separable sequences of components. If this separability cannot be easily performed then the benefit of the hierarchical structure in controlling the amount of information under consideration at any one time is lost.

The rectilinear bounding box is the easiest of the three to construct. The convex hull seems to be easier to compute (exactly) than the minimum-volume bounding box. Another advantage of convex hulls is that hulls of several structures can be easily aggregated into one larger containing hull. Because the minimum-volume bounding box does not properly contain all minimum-volume bounding boxes defined on its subsets, this type of aggregation cannot be done for bounding boxes. This is an important consideration when constructing a hierarchy of such approximations.

The bounding box approximation can be stored in constant
space regardless of the complexity of the underlying structure while the storage of the convex hull grows in direct proportion to this complexity. A bounding box also admits to a more efficient intersection algorithm. As noted by Rubin and Whitted [RubWhit,80], the standardization of the bounding box data structure allows the possibility of a hardware implementation of the bounding box intersection.

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. There is no 'best' approximation scheme over all possible point sets because each approximation element has its own relative merits and will perform better on some point sets than the others.

5.4 Hybrid Hierarchies

Some of the benefits of both would be obtained by using a hybrid hierarchy. This is a scheme in which various approximation elements including minimum-volume bounding boxes, three-dimensional convex hulls, and others may be used as hierarchy elements.

One way of structuring 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 encourage separability in doing interval sorting and to decrease the amount of work passed to the clipping routines.

A more systematic and efficient use of a hybrid hierarchy is to let the notion of 'tightness of fit' determine which bounding structure was to be used for a particular hierarchy element. This would be incorporated by calculating the convex hull and minimum-volume bounding box for the element to be approximated and calculating their respective volumes [LeeReg,80] [LeeReg,82a] [LeeReg,82b]. If the bounding box volume was within some factor of the convex hull volume, then the bounding box would be used. This would provide us with the benefits of the simplicity of the bounding box for an object that was essentially rectilinear.

The discussion of the beam/convex hull intersection in Section 5.2.3 implicitly presented the basis for another hybrid scheme. The elements of the outer sequence representation comprise a full spectrum of fits. The first element is a polyhedron defined on a constant number of vertices providing the simplicity of the bounding box. The last element of the sequence is the convex hull. Any one of the elements in this sequence could be chosen as the approximation element according to how well it fits the underlying structure.
In building an approximation element, the convex hull is first constructed and then the outer sequence representation of Section 5.2.3. is added. The volume of each element of the sequence can be compared with the volume of the convex hull. It is then easy to test at which element $Q$ in the sequence the volume increases past a pre-determined factor of the convex hull volume. All elements of the sequence past $Q$ can be discarded and their storage reclaimed. The rest of the sequence can be retained for performing the intersection testing as described in Theorem 5.4.

Asymptotically, a hierarchy of 'truncated' outer sequence representations takes no longer to build than the hierarchy of convex hulls. One of its advantages is that the hierarchy element used, and its storage, is related directly to how well it fits the underlying structure being approximated. Since the outer sequence representation of the chosen element is available, the efficient non-intersection detection described earlier can be performed. This advantage is that a crude initial test can eliminate it from further consideration and the amount of effort expended in performing the intersection only increases when the chance of a true intersection increases. This hybrid scheme also has the advantage that it can be applied to two dimensions; a polygonal face could be represented by its most appropriate outer sequence element.
6.1 Summary

This thesis presents a hierarchical approach to the hidden line/surface problem which uses approximation elements to package portions of the environment. Various ways of constructing and using such a hierarchical representation are proposed and discussed. Different convex approximation structures are examined as candidates for elements in this hierarchy.

We present our application area of 'computational acoustics' and motivate our research by relating the problems of hidden line/surface elimination to the problems of acoustics modeling beam-tracing. We generalize the notion of a viewing window to include the standard hidden-line viewing port, a pixel calculation in a hidden-surface raster scan, and a sound beam in the acoustics modeling.

We discuss the sequence of transformations necessary to preprocess an initial representation into the hierarchy described. It is shown how different initial representations can be transformed into the boundary representation and the associated face adjacency information. We also propose heuristics to use to segment automatically domain information for various application areas.
Algorithms are proposed which show how the approximation hierarchy can be used to solve efficiently the intersection, sorting and clipping subproblems of the hidden line/surface problem.

The two-dimensional convex hull and the minimum-area bounding rectangle are examined as candidates for planar approximations of environment faces and beam (viewing window) polygons. The three-dimensional convex hull, the rectilinear bounding box, and the minimum-volume bounding box are examined as candidates for the non-planar elements of the hierarchy. The complexities of constructing, storing and detecting the intersection of these candidates are presented and compared.

6.2 Conclusions

The hierarchical representation presented is useful for several applications in which it is desirable to limit the amount of information under consideration. The representation was designed to provide efficient techniques for performing acoustics beam tracing, but it has more general applicability. Although this can only be confirmed with an implementation, it seems that this representation is as easily used for the hidden surface elimination problem as for the acoustics beam tracing problem. This seems evident because the proposed algorithms provide polygonal output which can be used for graphical display or generation of sound reflections.
The hierarchy definition is general enough to include many different application areas. For example, the structures and algorithms described could be used with only slight modification to verify a proposed path for a robot on wheels. The path can be decomposed into a number of extrusions or beams. The algorithms presented can then be used to quickly determine whether or not any portion of the environment intersects any portion of the proposed path.

The transformations necessary to preprocess a given environment description into an approximation hierarchy are well-defined and tractable. It is unlikely that any domain independent preprocessing can be defined to transform all input into a segmented hierarchy. Domain dependent heuristics, such as the ones proposed in Chapter 3 will be necessary to obtain a good segmentation for input to the packaging routines.

The algorithms presented can exploit the hierarchical structure of our representation and can make use of recent results in computational geometry to do so efficiently. The structure of the algorithms is such that many computations can be done independently, suggesting the possibility of a parallel implementation of the window processing. Once the hierarchy is constructed, none of the algorithms modify it. Thus, for example, the recursive calls to handle passby beams could set up parallel processes for that purpose.
The hierarchical structure can use different convex polyhedral approximations as elements. There are no available candidates which are best overall in terms of construction, storage and intersection costs. Thus, a hybrid hierarchy can be used which will incorporate different approximation structures as hierarchy elements at different places in the hierarchy according to the structure's suitability.

6.3 Further Work

This thesis has been in many respects an interdisciplinary study of the hidden line/surface problem and its acoustics counterpart. The suggestions for further work must include the questions raised in each of the disciplines of graphics, geometric modeling and computer-aided design, computational geometry, and computational acoustics.

6.3.1 Graphics

Some related further work in graphics would be to examine the use of hierarchies to provide polygonal output as input to 'batched' shading, shadowing, transparency, and reflective surface processing.

Most hidden-surface shading algorithms normally work on a raster scan pixel calculation [NewSpr,79]. If the visible polygonal surfaces are identified before hand then a scan-line
conversion could be performed on them, calculating the shading value for each pixel. This could be thought of as 'batching' the hidden-surface elimination.

Shadows, as noted by [WeilAth,77], could easily be incorporated by performing the hidden-line elimination using the light source as the viewing position. The visible polygonal surfaces would then be the illuminated ones. Transparency effects could be incorporated by continuing a beam as a passby after it had hit a transparent surface. Reflective surfaces are dealt with exactly as acoustics sound beams are. The reflective surface is used to display the output of the reflected beam from the image source.

The hierarchical structure could be incorporated into animation. This could be done by moving the appropriate approximation element whenever a figure within it moved. As long as the contained objects did not change shape the containing approximation would not have to be recomputed.

Finally, further processing could be incorporated to precompute face priorities [SuSprSchu,74] to simplify the clipping. Then a group of faces within a beam could be ordered on the basis of their priority and the faces could be clipped out of the beam one at a time.
6.3.2 Geometric Modeling and Computer-Aided Design

The application area which we have been most closely associated with through our work is that of Computer-Aided Architectural Design. This subject area generates interesting problems for many disciplines. For example, a particular building design will be developed as a database of information. This database might then have the approximation structure which we have described superimposed upon it. The nature of the design process is dynamic; therefore, the database will almost certainly have to be modified. Under what circumstances can the hierarchical approximation structure corresponding to the design database be modified to effect the edited change rather than being recomputed? How can such editing be efficiently incorporated?

In the more general field of Computer-Aided Design, only recently have different representation schemes been categorized and compared [Req,80]. More extensive work is needed in examining different representations for representational adequacy, developing methods of extracting properties from different representations [LeeReq,82a] [LeeReq,82b], and developing transformations between equivalent representations.
6.3.3 **Computational Geometry**

Further work is needed on intersection and non-intersection detection for various structures. Are the inner and outer sequence representations [DobKirk,82] powerful enough to provide algorithms to match the best-known intersection detection results?

The techniques of computational complexity and the tools of computational geometry can provide important contributions in comparing and improving geometric processing in many application areas. For example, the Kirkpatrick [Kirk,80] algorithm for doing point location in planar subdivisions can be used to maintain the polygon comparison representation as described in Chapter 4. This search strategy can also be used to perform scan-conversion of a planar subdivision representing a group of polygonal surfaces to be displayed.

The theory of representation schemes being developed in geometric modeling [Req,80] can benefit from rigorous comparison of storage and processing costs. The discovery and application of efficient geometric algorithms for particular representations can influence the choice of representations and can make representation conversion more attractive.
6.3.4 Computational Acoustics

Our work will continue in implementing the hierarchical schemes described here and incorporating the results into our existing acoustics modeling system [Waldad, 82]. The adequacy of our model will be tested by modeling existing rooms and comparing the simulation results with actual room measurements.
References

[AhHopUll, 74]

[AvTouBha, 81]

[BaEaHen, 77]

[BaEaHen, 79]

[BallBro, 82]

[BoyDobDryGui, 82]

[ChazDob, 80]

[Cl, 76]

[DadKirkWal, 82]

[DobKirk, 82]

[DudaHart, 72]
[Ea, 77]

[Er, 80a]

[Er, 80b]

[EnHen, 77]

[ErLi, 75]

[ErLiSt, 75]

[ErTh, 79]

[FoliVan, 82]

[FourFuCarp, 82]

[FowLit, 79]
[FrSh, 75]

[GraYao, 81]

[GarJoPrepTar, 78]

[Kell, 58]

[Kirk, 81]

[KrokStrSor, 68]

[LeeReq, 80]

[LeeReq, 82a]

[LeeReq, 82b]

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

[NewSpr, 79]

[Pier, 81]

[PrepHong, 77]

[RedRub, 78]

[Req, 80]

[ReqVoel, 82]

[Rob, 63]

[RubWhit, 80]

[SchrAtal, 63]

[Sham, 78]

[ShamHoey, 76]
[Sl, 80]

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

[SuSprSch, 74]

[Til, 81]

[Wal, 79]

[Wal, 80]

[WalDad, 81]

[WalDad, 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.

[Weil, 80]

[WeilAth, 77]

[Win, 77]