

Computer Representations of Machined Parts for Automatic Inspection

by

John C. Fiala

Thesis submitted to the Faculty of the  
Virginia Polytechnic Institute and State University  
in partial fulfillment of the requirements for the degree of  
Master of Science  
in  
Electrical Engineering

APPROVED:

---

R. M. Haralick, Chairman

---

Paul Lapsa

---

John Roach

August 1985  
Blacksburg, Virginia

# Computer Representations of Machined Parts for Automatic Inspection

by

John C. Fiala

R. M. Haralick, Chairman

Electrical Engineering

(ABSTRACT)

A comparison is made between octrees, which are regular decompositions of volumes, and new, irregular decompositions called R-trees. This comparison is made within the context of the volume intersection problems that might be associated with an automatic inspection system. The results show that the irregular decomposition is independent of object position and can provide a more space efficient encoding for certain shapes. However, detecting intersections between R-trees requires an algorithm of greater complexity due to the irregularity of the decomposition.

Algorithms are given for obtaining tree decompositions from a hierarchic relational model of a volume. Among these algorithms is a procedure for finding the minimal enclosing rectangular parallelepiped of a boundary representation, and a generalization of the point-in-polygon algorithm to boundaries on curved surfaces. Both of these algorithms have computation times that are proportional to the total number of components of the boundary's representation.

## ACKNOWLEDGEMENTS

First of all, I would like to acknowledge Martin Marietta Corporation which funded, in part, the research for this thesis. I thank all the members of the Martin Marietta research group at the Spatial Data Analysis Lab who contributed so many ideas to this thesis. In particular I would like to thank

, and . A special thanks goes to , who gave me the opportunity to do this research and made outstanding contributions to it.

In addition to the Martin Marietta group, I would like to express my appreciation for the assistance of , Lord of VAX2, and , GIPSY guru extraordinaire, with SDA computing facilities.

I am very grateful to Paul Lapsa and John Roach for serving on my committee.

Finally, I thank my family and friends for the support they have provided throughout this endeavor.

## TABLE OF CONTENTS

Chapter 1 - Introduction . . . . .	1
Chapter 2 - Literature Review . . . . .	6
CSG and Boundary Representation . . . . .	7
More Recent Modeling Schemes . . . . .	9
Tree Decompositions of Volume . . . . .	11
Chapter 3 - A Hierarchic Relational Model . . . . .	17
Spatial Data Structures . . . . .	25
Object Representation . . . . .	26
The Use of Frames . . . . .	34
Applicability to Collision Checking . . . . .	35
Chapter 4 - The Rectangle-tree . . . . .	38
Description of the Rectangle-tree . . . . .	39
Experimental Comparison to Quadtree Structures . . . . .	43
Chapter 5 - The Rectangular Parallelepiped-tree . . . . .	58
Experimental Comparison to Octree Structures . . . . .	60
Chapter 6 - Obtaining R-trees . . . . .	68
Notation . . . . .	69
The Outside RPP of a Part . . . . .	73
Table of Contents	iv

Point-in-Boundary Problem . . . . .	86
A Discretized Representation . . . . .	96
The Inside RPP . . . . .	101
Tree Building . . . . .	106
<b>Chapter 7 - Operations on R-trees and Octrees . . . . .</b>	<b>109</b>
Translation . . . . .	109
Rotation . . . . .	110
Interference Detection . . . . .	111
<b>Chapter 8 - Conclusions . . . . .</b>	<b>120</b>
<b>Bibliography . . . . .</b>	<b>122</b>
<b>Vita . . . . .</b>	<b>129</b>

## LIST OF ILLUSTRATIONS

Figure 1. Automatic Inspection . . . . .	2
Figure 2. Octree Construction . . . . .	14
Figure 3. Hierarchic Relational Model . . . . .	18
Figure 4. Right-handed Boundary . . . . .	28
Figure 5. Twisted, Planar Surface Patch . . . . .	31
Figure 6. Arc Ambiguity . . . . .	33
Figure 7. Frame Relationships of Model . . . . .	36
Figure 8. Rectangle-tree Decomposition . . . . .	41
Figure 9. Children at Level 2 Node . . . . .	42
Figure 10. Generalized Tree Structure . . . . .	44
Figure 11. Rectangle-tree Structure . . . . .	45
Figure 12. Quadtree Structure . . . . .	47
Figure 13. Random Generation of Polygonal Regions . . . . .	48
Figure 14. Rectangle Intersection Algorithm for Quadtree	50
Figure 15. Rectangle Intersection Algorithm for Rectangle-tree . . . . .	51
Figure 16. Line Images . . . . .	57
Figure 17. RPP-tree Decomposition . . . . .	59
Figure 18. Four-sided Polyhedron . . . . .	61
Figure 19. Quadrilateral Cylinder . . . . .	63
Figure 20. Quadrilateral Cylinder at Corser Resolution	64
Figure 21. Cubic Shell . . . . .	66
Figure 22. Point-in-Polygon Solution . . . . .	87
Figure 23. Indistinguishable Boundary Intersections . . . . .	90
Figure 24. Point-to-Boundary Connections . . . . .	91
List of Illustrations	vi

Figure 25. Planar Conic Arc . . . . . 94  
Figure 26. Point-in-Boundary Solution . . . . . 97  
Figure 27. Single Piece Example of Discretization . 100  
Figure 28. Possible Inside Rectangles. . . . . 103  
Figure 29. Row-by-Row Development of Inside Rectangle 105  
Figure 30. Octrees Intersection Algorithm . . . . . 112  
Figure 31. R-trees Intersection Algorithm . . . . . 115  
Figure 32. Small Polyhedron . . . . . 117  
Figure 33. Shifted Small Polyhedron . . . . . 118

LIST OF TABLES

Table 1.	Runs on Complete Quadrees	. . . . .	52
Table 2.	Black Quadtree Comparison	. . . . .	53
Table 3.	Modified Black Quadtree Comparison	. . . . .	54

## CHAPTER 1 - INTRODUCTION

Consider the problem of inspecting a large machined part as an example of a demanding automation task. This scenario is depicted in Figure 1. Two robot arms, equipped with cameras for vision and an assortment of interchangeable end effectors for taking measurements, are to inspect a machined part, such as an aircraft bulkhead, the dimensions of which are critical. The bulkhead moves along a track in the x-direction to provide the arms with access to its entire length.

An aircraft bulkhead, being so large, may require over 1,100 inspection measurements, traditionally done by hand, many of which are vital to the proper construction of the aircraft. Therefore, such an automation system would be desirable; how can it be built?

There are many facets to a robotic system such as this, including robot accuracy, coordination of multiple actors, and real-time control. But one of the central components of the system will be the representation of the environment. In order to do path planning and collision checking, the system must have a "good" representation of the objects in its environment and of their relative positions and orientations. In order to recognize objects and their positions visually, the system must have descriptions of what objects look like.

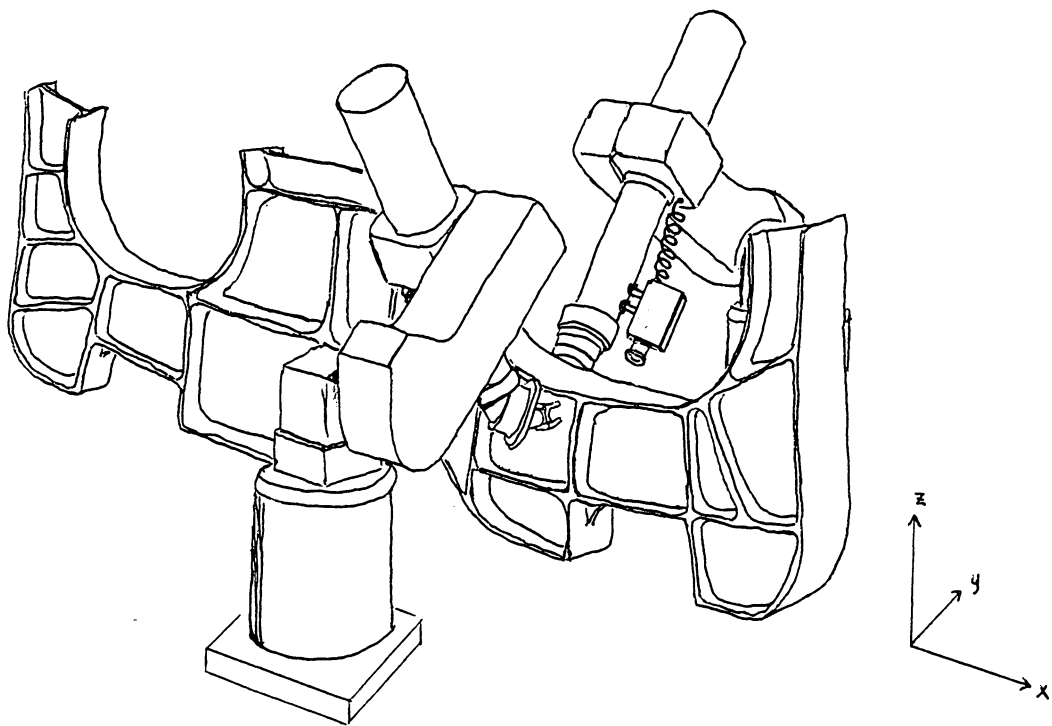


Figure 1. Automatic Inspection.

Likewise, an accurate representation of the bulkhead must be available so that the system can make decisions about the measurements being taken. Therefore, the system must have a database to provide the representations necessary for its operations. This database may contain considerable redundancy, different tasks being more efficiently executed using different data structures, but the governing criterion is that the various functions of the system be carried out rapidly and in a coordinated manner.

Ideally, the system should be able to obtain its original information from the Computer Aided Design (CAD) system on which the part was designed. This means that the system must have some way of converting the CAD representation into other, more appropriate representations. Requicha and Voelcker, solid modeling researchers at the University of Rochester, recognized in a recent paper the need to have multiple representations of an object, but remarked that there is a lack of literature on the algorithms that do the conversions between representations [46].

The purpose of this thesis is to provide some information on the process of converting from an exact, boundary representation of a solid to a tree data structure representation, e.g. an octree. In addition, this thesis will explore what is meant by a "good" representation in the context of volume intersection and robot collision checking problems by comparing a regular and an irregular decomposition of volumes.

Much of the work contained in this document is based on work done previously by many researchers associated with the Spatial Data Analysis Laboratory [21,22,31,60,67,72], but this is the first time it has appeared in published form. For this reason, and because changes and additions have been made, the previous work is included to make the discussion complete. In particular, Chapter 3 is a description of the hierarchical model of the bulkhead presented in [60]. It is presented again here since this is the boundary representation on which this thesis is based.

Note that the hierarchical model is a relational structure of the form described by Shapiro and Haralick in [58]. Thus the representation was implemented using the Pascal-based Spatial Information System developed by S. N. Engineer [10], and P. D. Vaidya [65], which implements this type of spatial data structure.

Chapters 4 and 5 relate to a preliminary analysis of an alternate data structure to the octree called an R-tree. This comparison of the two data structures is similar to an earlier two dimensional analysis made in [21]. These experiments were all done through the GIPSY system using the RATFOR (RATional FORTRAN) language [28,29].

The conversion algorithms described in Chapter 6 were also implemented and tested through GIPSY. Chapter 6 examines the problem of finding a minimal enclosing rectangular parallelepiped of a part given its boundary representation

and discusses algorithms for building the tree data structures.

Finally, Chapter 7 presents some results involving operations on irregular and regular tree decompositions, i.e. R-trees and octrees. The experiments here were also done using GIPSY.

But first, to clarify some of the terminology used in this introduction, and examine the related work being done elsewhere, a brief review of the literature will be made in the next chapter.

## CHAPTER 2 - LITERATURE REVIEW

A wide variety of techniques for representing solids has appeared in the literature. Perhaps many other techniques are employed in commercial systems which, due to their proprietary nature, have not been published. This chapter will review some of the most popular and most current of the techniques available in the literature, with particular emphasis on those representations closely related to this thesis. For the purposes of this discussion it will be assumed that representations are of known solids, i.e. it is not necessary to derive the representation from measurements of the object. For a survey more closely related to obtaining representations from measurements, see [1].

Ideally, the representations of an object needed for any operation in a vision or robotic system could be obtained from the original CAD design. Therefore, the CAD representation may play a central role in the set of representations for an object. Requicha [44], and Requicha and Voelcker [45] have presented surveys of solid modeling as related to Computer-Aided Design, and in [46] make this very point about CAD's central role. Ballard and Brown, in their text on computer vision [5], present a slightly different overview of solid modeling.

## CSG and Boundary Representation

By far, the two most popular techniques for representing solids are Constructive Solid Geometry (CSG) and boundary representation. At least one of these techniques is the central approach of almost every commercial CAD system today [46]. In CSG, complex solids are represented by the combination (union, intersection, deletion, etc.) of primitive solids. Primitives are usually simple shapes such as cuboids, cylinders, cones, and spheres, and are often represented as quadric equations with bounding inequality constraints [36,68]. A boundary representation of a solid is a set of surface patches, each patch, called a face, usually being defined by a set of bounding edges. The union of the faces defines a simply closed surface and, by the intuitive Jordan theorem [2], divides three-space into two parts, an interior which is the solid, and an exterior which is empty space. Obviously, these two representations are related and could be used together, a primitive of a CSG scheme having a boundary representation.

The principle difference between the two techniques is that in a CSG representation of a complex object all edges do not appear explicitly, some being created by the intersection of primitives, whereas in a boundary representation they do. Thus, though the simplicity of a CSG representation is convenient for graphics rendering [36,48,68] and user in-

put [14,44], the boundary representation with its explicitly defined edges and surfaces is more appropriate for computer vision [34].

If a boundary representation is to model a solid then a decision must be made regarding the representation of surfaces and edges. The simplest choice is to restrict all surfaces to be planar and all edges to be straight line segments. This approach, being simple, has been widely used [8,64,69]. Unfortunately, it provides an accurate description of curved objects only at the cost of having a large number of planar surface pieces. Since a simple curved surface such as an ellipse or cone can be described with a single, second-order equation, quadric surface boundaries have been used [36,68,55] instead of these polyhedral approximations. The principle disadvantage of using quadrics is their algebraic complexity as compared with planar surfaces. For example, to determine extrema one must take derivatives [72] instead of looking solely at the endpoints of edges. Likewise, the problem of computing surface intersections becomes difficult [32,55]. In addition, the problem of having an arbitrary boundary for a quadric surface has not been addressed in the literature. ( See Chapter 6.)

Another approach to surface representation, one that has been lauded for CAD systems, is that of parametric basis functions. In this approach, the surface is an element in the span of a set of basis functions, often called blending

functions, which are functions of two parametric variables. The surface, then, being a linear combination of the basis functions, is generally determined by a vector of control points. This type of surface description is useful for complicated surfaces not amenable to planar or quadric representations. Forrest [15], and Newman and Sproull [40], present reviews of the basic concepts involved. Goldman [18] shows how two of the most popular forms of basis functions, Bezier and B-splines, arise naturally from probability theory.

Of course, the same general techniques discussed here with regard to surfaces can be applied to curves as well. In this case, a curve is an element in the span of a set of functions of one parametric variable.

### More Recent Modeling Schemes

Although CSG and the boundary representation are the most popular forms in solid modeling design systems, many other representations have been proposed. Most of these representations are related in some fashion to the previous two. However a distinction should be made between a representation of an object and a description of an object. Vision systems often are not concerned with the "exact" shape of an object but only want to know what the object "looks like". Thus, although many descriptions of objects have been proposed, they will not be discussed here since they do not contain

enough to fully represent the object, i.e. it is not possible to construct other representations from this description.

Lin and Fu [34] presented an interesting solid modeling scheme which is essentially a boundary representation with the addition of a context-free grammar to describe how the surfaces are put together. Thus, an object belongs to a context-free language and could possibly be used in some form of syntactic pattern recognition.

Faugeras and Ponce [11] have proposed Prism trees for 3-D object representation, which are an extension of Ballard's 2-D Strip trees [4]. A Prism tree is a hierarchical structure where each succeeding level represents a closer approximation to the object. The nodes of the tree are prisms which enclose some portion of the boundary. Therefore a Prism tree represents a solid by its boundary as did the boundary representation. This representation is useful for determining when two surfaces intersect, but has the disadvantage of involving the intersection of triangular prisms in arbitrary orientations.

Another interesting type of representation is given by O'Rourke and Badler [41]. This idea involves the decomposition of a solid into a set of possibly overlapping spheres. Because the union of the spheres ideally gives the entire object, this method is closely related to the CSG representations. One disadvantage of this representation is that it may require a very large number of spheres to represent a

complex object to the desired accuracy. However, it appears that determining intersections between spheres may provide fairly efficient interference detection [23].

The generalization of Blum's 2-D symmetric (or medial) axis transformation [6] to 3-D has been presented by Nackman and Pizer [39]. The symmetric axis transform decomposes an object into a central surface called a skeleton, each point of which has a "radius" value which specifies the distance to the object's boundary. One can imagine a sphere being centered at each of these skeletal points, having the specified radius. Thus the symmetric axis transformation is essentially an infinite version of the sphere decomposition of O'Rourke and Badler.

### Tree Decompositions of Volume

The idea of representing an object in a hierarchical tree structure, as in the Prism tree, where succeeding levels represent levels of higher resolution, can provide efficiency both in access times and storage requirements. Also, the idea of decomposing an object into a set of primitives all of which are of the same simple form, as in the case of the sphere decomposition, offers the advantage of having to deal only with simple objects. These two ideas form the basis of another solid representation scheme that has received much attention in the recent literature [3,17,25,38]. In this

representation, commonly referred to as an octree (or oct-tree) representation, the primitives are cuboids whose faces are all parallel to the rectangular coordinate axes of the world (or universe). The concept of a world coordinate system simply provides a reference so that the octrees of all objects will be in the same orientation. Since the cuboids at any level of the tree are disjoint and represent a higher resolution, i.e. are a decomposition of the parent cuboids, octrees can be thought of as a constructive solid geometry representation where the only operation allowed on the primitives (cuboids) is "gluing".

A thorough survey of data structures related to the octree has been made by Samet [54]. His overview emphasizes two-dimensional versions of the octree, commonly called quadtrees (or quad-trees). The reason for this is that much use has been made of quadtrees in computer graphics and image processing. In fact, one of the earliest uses of a quadtree-like decomposition was made by Warnock [66] in his hidden surface removal algorithm. Of particular interest is the study of quadtrees produced by Hunter and Steiglitz [24]. Their paper gives the basic techniques for performing operations on quadtrees such as union and intersection. These techniques are extendable to octrees as is most of the work related to quadtrees. The data structure can also be generalized to K-dimensions [26,70,71]. However, since this thesis focuses on

solid modeling, three-dimensional structures will be of primary interest.

An octree is constructed as follows. Begin with a large cube that contains the entire work space (or universe) of all objects, including the object to be represented by this tree. This cube is the root node of all octrees of all objects. The edges of this cube define the directions of the world coordinate axes. Now, divide the cube in half in each of the three coordinate directions generating eight "octants", Figure 2. These octants are the children of the root node. If an octant contains none of the volume of the object it is labeled "empty". If an octant is completely filled by part of the object's volume, it is labeled "full". Octants which are "full" or "empty" are not subdivided further, but "mixed" octants are recursively subdivided in eight parts until either no "mixed" octants are generated or the desired resolution is reached.

From this description it can be seen that an octree is a regular decomposition of a volume, in the sense that divisions are made at predetermined, regular intervals. Any level of the tree represents a known, evenly spaced subdivision of the original cube. Klinger and Dyer [27] emphasize that regular decompositions of this form have many advantages, one of which is that any geographical part of the image may be rapidly accessed. Also, since the form of the decomposition is predetermined, the tree can be encoded in a linear array

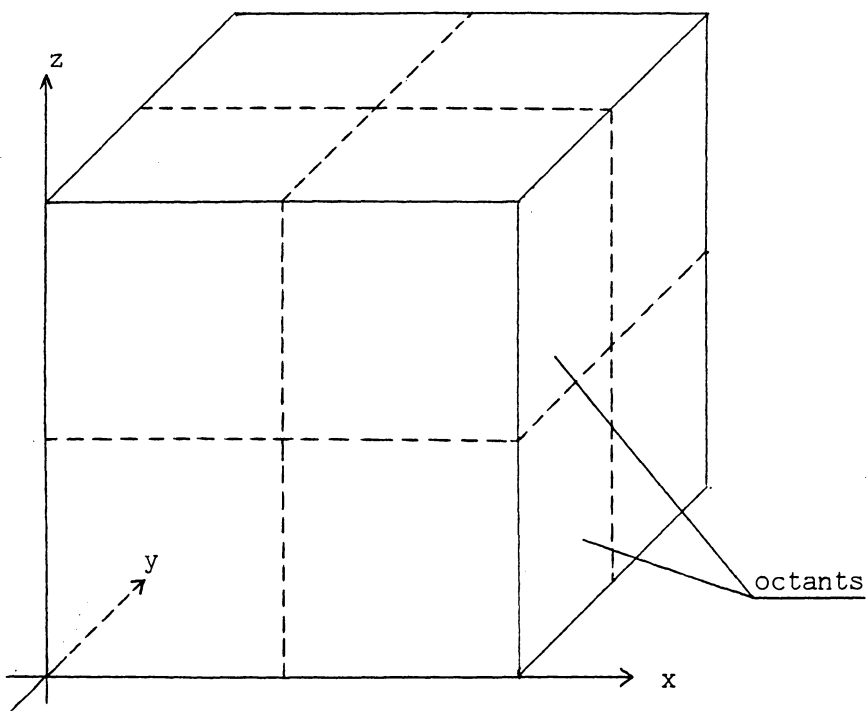


Figure 2. Octree Construction

format to save space as proposed by Gargantini [16,17] and Tamminen [63]. Unfortunately, as pointed out by Ahuja and Nash [3], octree representations are not constructed with respect to the object itself but with respect to a world coordinate system. For this reason, the representation is not independent of the object's position. This presents two problems. One, if an object is translated a new octree must be constructed [3,25,38]. And two, the space required for an octree representation varies greatly with the object's position relative to the universe cube [33].

The reader may have noted that one nice feature of a symmetric axis transform representation is that it is independent of the object's position. Is it possible then to construct a representation that has this same independence of position, but retains the simple decomposition primitive of a box whose faces are parallel to a world coordinate system? The answer is "yes". The desired representation would correspond to an irregular decomposition based on the object's shape. Rubin and Whitted [49] recognized that an irregular decomposition of this form could improve both time and space efficiency in computer graphics applications, but the idea has not been pursued further. This type of representation is explored in detail in Chapters 4, 5, and 7.

First, the solid model from which tree structures can be constructed will be examined. This representation, which is

a special combination of CSG and boundary representations, will be described in the next chapter.

## CHAPTER 3 - A HIERARCHIC RELATIONAL MODEL

A relational model has been chosen as a basis for object representation primarily because of its usefulness in vision [59]. Matching problems related to object recognition can be reduced to the problem of determining homomorphisms between graphs in this relational approach [58]. Shapiro and Haralick have given a relational model for an aircraft bulkhead in [60]. This model forms the basis for subsequent developments in this thesis.

A large machined object can be represented by a relational model which is a hierarchic composition of spatial data structures. The model for the robotic inspection system is given in its entirety in Figure 3. The structures of principle interest are the ones at or below the object level since this thesis is mainly concerned with the representation of objects. The world level describes the positions and relationships of objects in the world, while the object and lower levels actually represent the objects. To examine how objects are represented in this model, the basic structure, the spatial data structure, should first be formalized.

## I. WORLD LEVEL

World SDS

## Objects Relation

Object	Type	Description	Transformation
name	value	pointer to Object SDS	pointer to matrix

## Objects Connection Relation

Object1	Object2	Type of Connection	Connection
name	name	value	pointer to Connection SDS

## Current Spatial Relationships of Objects Relation

Object1	Object2	Type of Relationship
name	name	value

## Attribute-Value Table

x-deviation	value
y-deviation	value
z-deviation	value
Containing Rectangular Parallelepiped	pointer to Part SDS

Figure 3. Hierarchic Relational Model

## II. OBJECT LEVEL

Object SDS

## Parts Relation

Part	Type	Description	Transformation
name	value	pointer to Part SDS	pointer to matrix

## Parts Connection Relation

Part1	Part2	Type of Connection	Connection
name	name	value	pointer to Connection SDS

## Distinguished Point Relation

Point	Location	Parts List	Quality
name	point in object coordinates	name list	value

## Attribute-Value Table

Containing Rectangular Parallelepiped	pointer to Part SDS
Height	mean deviation
Width	mean deviation
Depth	mean deviation
Weight	mean deviation

Figure 3. Hierarchic Relational Model (continued)

Connection SDS (type 1 connection)

Attribute-Value Table (varies with connection type)

Distinguished Point	name in owner object
Nature	list of values

Touching Boundaries Relation

Boundary	Description	Surface1	Visible1	Surface2	Visible2
name	pointer to Boundary SDS	touching surface name w.r.t. part1	value	touching surface name w.r.t. part2	value

Touching Edges Relation

Edge	Description	Boundary	Surface1	Surface2	Angle	Fillet	Visible
name	pointer to Arc SDS	name	important adjacent surface w.r.t. part1 of object	important adj. surface w.r.t. part2	mean dev.	ptr to Fillet SDS	value

## III. PART LEVEL

Fillet SDS

Attribute-Value Table

Defining Piece	pointer to Piece SDS
Containing Rectangular Parallelepiped	pointer to Part SDS

Figure 3. Hierarchic Relational Model (continued)

Part SDS

## Surfaces Relation

Surface	Type	Description	Transformation
name	value	pointer to Surface SDS	pointer to matrix

## Edge/Surface Topology Relation

Edge	Vertex1	Vertex2	Surface Left	Arc of Left	Surface Right	Arc of Right	Angle
name	name	name	name	ptr to arc	name	ptr to arc	mean dev.

## Vertex Relation

Vertex	Location	Edge List
name	point in part coordinates	name list

## Holes Relation

Hole	Surface	Radius	Center	Depth	Axis	Transformation	Volume
name	name	mean dev	point	mean dev	ptr to arc	ptr to matrix	pointer to Part SDS

## Attribute-Value Table

Containing Rectangular Parallelepiped	pointer to Part SDS
Length	mean deviation
Width	mean deviation
Depth	mean deviation

Figure 3. Hierarchic Relational Model (continued)

## IV. SURFACE/ARC LEVEL

Surface SDS

## Attribute-Value Table

Area	mean deviation
Shape	value
Boundary	pointer to Boundary SDS

## Pieces Relation

Piece name	Description	Transformation
	pointer to Piece SDS	pointer to matrix

Piece SDS

## Attribute-Value Table

Primary Description:  $ax^2 + by^2 + cz^2 + dx + ey + fz + g = 0$

Coefficient of $x^2$ term	value
Coefficient of $y^2$ term	value
Coefficient of $z^2$ term	value
Coefficient of x term	value
Coefficient of y term	value
Coefficient of z term	value
Constant term	value
x-interval	(value,value)
y-interval	(value,value)
z-interval	(value,value)
Intervals Exact	boolean
Boundary	pointer to Boundary SDS

Figure 3. Hierarchic Relational Model (continued)

Secondary Description: Linear Combination of Parametric Basis Functions

n= dimension of space	value
Parameter u interval	(value,value)
Parameter v interval	(value,value)
Basis	pointer to Basis SDS

Coefficient Relation (length = n)

x-coefficient	y-coefficient	z-coefficient	w-coefficient
value	value	value	value

Boundary SDS

Arc List Sequence Relation

Arc	Description	Transformation
name	ptr to Arc SDS	pointer to matrix

Arc SDS

Attribute-Value Table

Primary Description:  $ax^2 + by^2 + cx + dy + f = 0$

Coefficient of $x^2$ term	value
Coefficient of $y^2$ term	value
Coefficient of x term	value
Coefficient of y term	value
Constant term	value
x-interval	(value,value)
y-interval	(value,value)

Figure 3. Hierarchic Relational Model (continued)

Secondary Description: Linear Combination of Parametric Basis Functions

n= dimension of space	value
Parameter t interval	(value,value)
Basis	pointer to Basis SDS

Coefficient Relation

x-coefficient	y-coefficient	z-coefficient	w-coefficient
value	value	value	value

Basis SDS

Attribute-Value Table

Number of Basis Functions	value
---------------------------	-------

Functions Relation

Basis Function

function definition
---------------------

Figure 3. Hierarchic Relational Model (continued)

## Spatial Data Structures

A spatial data structure [58], also called a relational data structure [59], is a set of relations  $\{ R_1, R_2, \dots, R_K \}$ . Each relation  $R_i$ ,  $i=1, \dots, K$ , may be either an unlabeled relation,

$$R_i \subseteq S_{i1} \times S_{i2} \times \dots \times S_{iN_i},$$

where the  $S_{ij}$ ,  $j=1, \dots, N_i$ , are the sets on which the relation is defined, or a labeled relation,

$$R_i \subseteq L_i \times S_{i1} \times S_{i2} \times \dots \times S_{iN_i},$$

where  $L_i$  is a set of labels and the  $S_{ij}$ ,  $j=1, \dots, N_i$ , are as before. A relation  $R_i$  is therefore a set of  $N_i$ -tuples or  $(N_i + 1)$ -tuples depending on whether it is unlabeled or labeled, respectively. The elements of the  $S_{ij}$  may be either atoms, e.g. integers, real numbers, or character strings, or spatial data structures (SDS's). The elements of the label set  $L_i$  may be names or, to form an ordered relation, whole numbers. The hierarchy of the model is obtained by having lower level SDS's as elements of the sets of the relations of higher level SDS's. For example, object SDS's appear in the objects relation of the world SDS. In Figure 3 SDS's are

described by the appropriate set of relations, and each relation is given by the form of its n-tuples.

## Object Representation

An object is represented in this model as a set of parts. Each part is a solid of relatively simple form, and thus, this decomposition of complex objects facilitates the encoding of the representation. Note the similarity of this idea to the constructive solid geometry approach. The part representation is the primary concern of this thesis for reasons to be given later in this chapter.

The part SDS consists of five relations. The surfaces relation, the edge/surface topology relation, and the vertex relation form a basic boundary representation of the part. Instead of requiring this boundary representation to include holes in the part, a requirement that would greatly increase the complexity of the boundary, the holes relation is included to describe them. The holes are thus represented as volumes, i.e. part SDS's, and are combined with the boundary representation of the part via the constructive solid geometry operation of deletion. The fifth relation of the part SDS is the attribute-value table relation. The attribute-value table of an SDS is an unlabeled relation that contains only one n-tuple and gives some overall properties of the entity described by the SDS. For a part, the attribute-value

table gives an enclosing rectangular parallelepiped that can be used to roughly describe the part.

A part is represented primarily by its surfaces and edges. Each surface is given by a set of pieces (to facilitate encoding.) The union of all of these pieces, or surface patches, represents the entire surface. The attribute-value table for the surface SDS includes a boundary of the surface. The boundary is represented by a boundary SDS which consists solely of an arc list sequence relation. This relation is an ordered, labeled relation that lists the arcs that compose the boundary. These arcs are ordered so that if the fingers of the right hand follow the boundary arc ordering, the thumb will point outward from the part surface, as in Figure 4. The arcs of the surface boundary are the arcs that appear in the edge/surface topology relation of the part SDS.

The piece SDS provides a mathematical description of a surface patch. The attribute-value table for a piece contains the parameters for two different types of descriptions. Both descriptions may exist for a given piece, or maybe only one will apply.

The primary description for a piece is a quadric surface equation,

$$Ax^2 + By^2 + Cz^2 + Dx + Ey + Ez + G = 0, \quad (1)$$

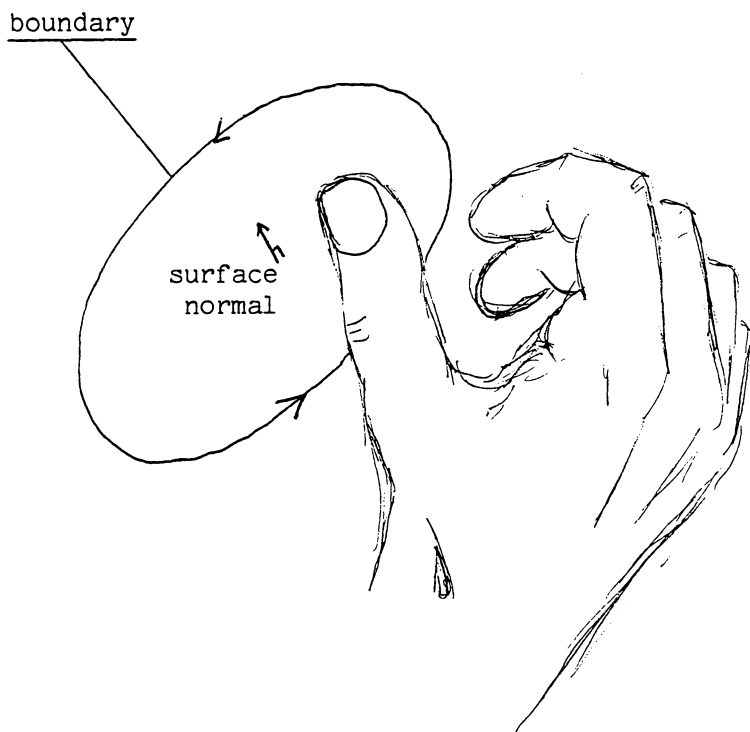


Figure 4. Right-handed Boundary

along with a boundary of arcs that lie on this surface and define the limits of the piece. A bounding rectangular parallelepiped of the piece is given by x-, y-, and z-intervals. When this box exactly delimits the piece, i.e. the piece is a planar, rectangular surface patch, the "intervals exact" flag is set so that the boundary need not be examined in detail. The boundary of a piece, like the boundary of a surface, has a direction. That direction is defined to be a right-handed one, as before, or equivalently, the direction such that when walking around the boundary on the outside of the part the surface is to the left. In addition, each arc of the boundary is directed from one endpoint to the other in a manner that corresponds to the direction given by the boundary.

The normal vector to a piece described by (1) is defined to be

$$\begin{aligned} \mathbf{n} &= \nabla ( Ax^2 + By^2 + Cz^2 + Dx + Ey + Fz + G ) \\ &= ( 2Ax + D )\mathbf{x} + ( 2By + E )\mathbf{y} + ( 2Cz + F )\mathbf{z}, \end{aligned}$$

where  $\mathbf{x}$ ,  $\mathbf{y}$ , and  $\mathbf{z}$  are the orthonormal coordinate vectors. This vector points outward from the part.

The secondary description of a piece represents the surface patch as a linear combination of parametric basis functions,

$$p(u,v) = \sum_{i=1}^n a_i \beta_i(u,v),$$

where  $n$  is the dimension of the representation space, the  $a_i$ ,  $i=1, \dots, n$ , are the coefficient vectors, and the  $\beta_i(u,v)$ ,  $i=1, \dots, n$ , are the basis functions parametric in  $u$  and  $v$ .

The parameters  $u$  and  $v$  range over the values specified in the attribute-value table of the piece. The coefficient vectors  $a_i$ ,  $i=1, \dots, n$ , are stored in the ordered coefficient relation. And the basis functions are defined in a basis SDS.

A wide variety of standard methods of surface representation fit into this format, including B-splines, Bezier surfaces, and Coons patches [19,67,40,15]. As an example, consider Bezier surfaces of order 1.

$$n = 4$$

$$u\text{-interval} = v\text{-interval} = (0,1)$$

$$\text{basis} = \{ (1-u)(1-v), (1-u)v, u(1-v), uv \}$$

These basis functions provide an easy way to describe a surface patch taken from a twisted plane. The coefficient vectors for the twisted plane piece in Figure 5 are

$$a = \begin{bmatrix} a_x \\ a_y \\ a_z \\ a_w \end{bmatrix} = \begin{bmatrix} 0 & 4 & 5 & 1 \\ 0 & 2.5 & 0 & 2.5 \\ 0 & 5 & 5 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}.$$

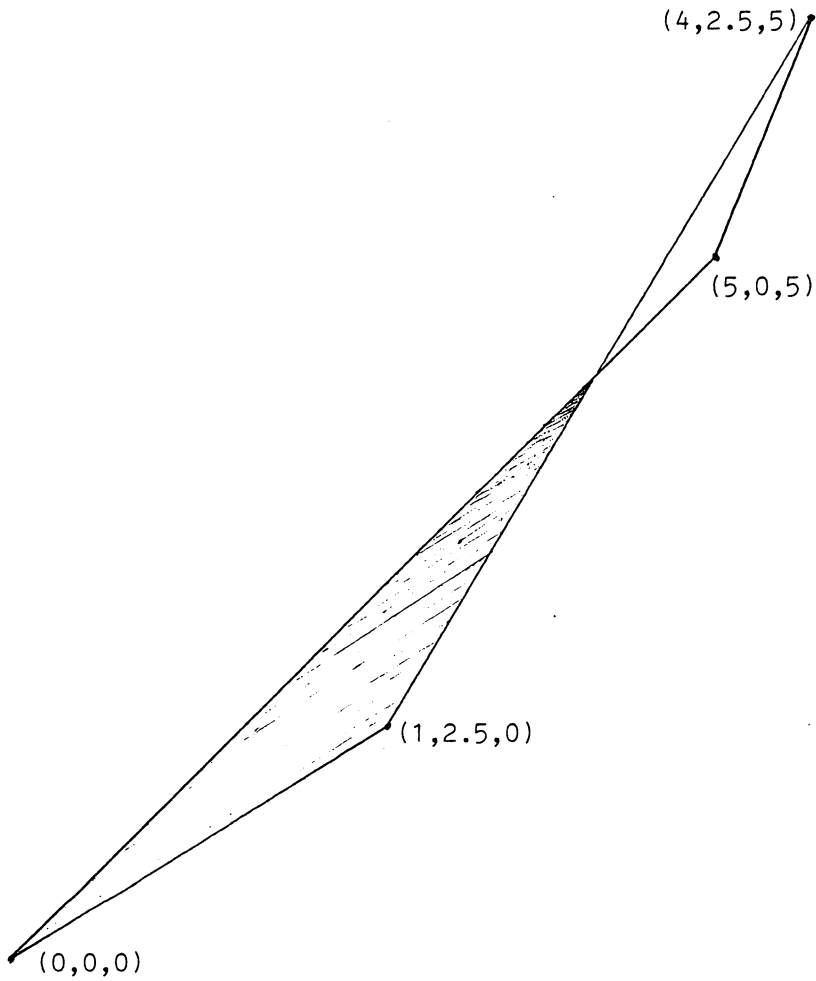


Figure 5. Twisted, Planar Surface Patch

Arcs which appear in boundaries of the model are represented by arc SDS's. The arc SDS, like the piece SDS, provides mathematical representations in either of two forms. The parameters of these forms are given in the attribute-value table.

The primary description represents an arc as a segment of a quadratic equation,

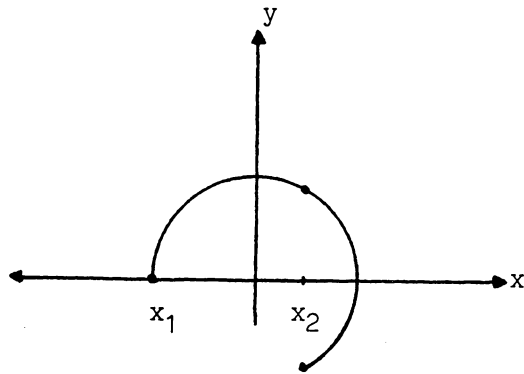
$$Ax^2 + By^2 + Cx + Dy + F = 0, \quad (2)$$

The form of this description requires that the arc being represented turn no more than  $180^\circ$ , since ambiguities arise otherwise. See Figure 6. Also, since arcs are directed, the primary description must provide an implied direction to the arc. This direction, for an x-interval of  $(x_1, x_2)$  and (2), is assumed to be from the endpoint  $(x_1, f(x_1))$  to the endpoint  $(x_2, f(x_2))$ , where  $f(x)$  is the implicit function  $y=f(x)$  of (2).

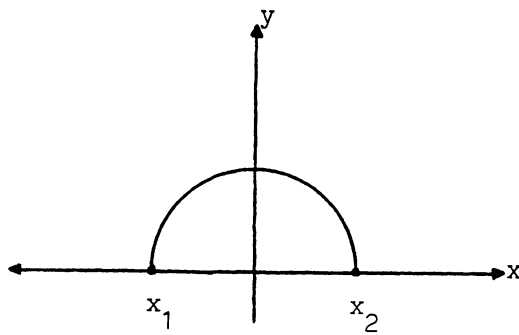
As in the piece SDS, the secondary description of an arc is a linear combination of parametric basis functions,

$$p(t) = \sum_{i=1}^n a_i \beta_i(t).$$

Associated with this description is the coefficient relation which gives the values of the  $a_i$ ,  $i=1, \dots, n$ . The basis functions  $\beta_i(t)$ ,  $i=1, \dots, n$ , are given by a basis SDS as in



(a)



(b)

Figure 6. Arc Ambiguity: Endpoint at  $x_2$  is ambiguous in (a) but not in (b).

the piece case. For this secondary description, the direction of the arc is defined to be from the point  $p(t_1)$  to the point  $p(t_2)$  for a  $t$ -interval of  $(t_1, t_2)$ .

### The Use of Frames

It can be seen from this discussion that this relational model provides a combination of constructive solid geometry and boundary representations. The part level of the model is primarily a boundary representation, while at higher levels these simple solids are constructed into more complex objects. One of the central features of both of these techniques is the use of homogeneous coordinate transformations to describe the relationships. This allows entities to be described in the most convenient position and orientation, and then "transformed" into the desired position. Also, entities may be "shared" through the use of different transformations for different parts of the object. All transformations in the model are rigid body transformations conforming to the conventions of Paul [42].

Each level of the model has its own coordinate system that is related to other coordinate systems via these homogeneous coordinate transformations. To simplify discussions, the coordinate system of a level will be referred to as a frame. The notation for a frame of a level will be  $\text{Frame}(\text{level})$ . Thus

the relationship of frames of the relational model can be depicted as in Figure 7.

In the figure, the transformation that relates the world coordinate system to the object coordinate system is  $T_1$ . To relate frames that differ by more than one level, multiply the intermediate level transformations together. For example, the transformation that relates Frame(world) to Frame(piece) is  $T = T_1 T_2 T_3 T_4$ .

### Applicability to Collision Checking

In concluding this discussion of the hierarchic relational model, the reader should note the complexity of this representation. Although it provides an exact representation in a form amenable to computer vision, this model would be inefficient for determining intersections between objects. This type of collision checking problem requires a data structure designed so that collision checking can be done rapidly.

An efficient data structure for collision checking is the octree [38]. Obviously, encoding an object as large as an aircraft bulkhead to a reasonable degree of resolution would require a huge amount of space. But the fact that the bulkhead will be translated in space, requiring that the octree be reconstructed on the fly, makes this approach untractable. A more reasonable approach involves two fundamental changes. One, do not encode the entire bulkhead in one

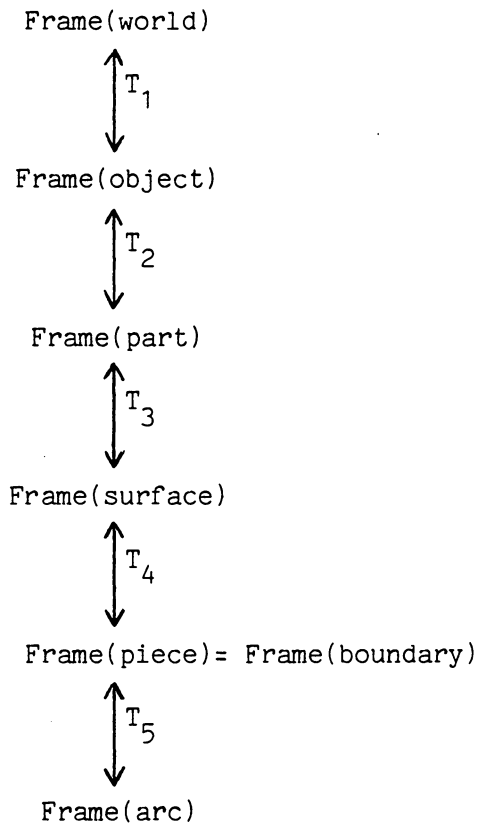


Figure 7. Frame Relationships of Model

structure but encode each part in its own hierarchic tree structure. This is acceptable since each robot arm will only be moving in the vicinity of a small number of parts at any given time. Two, use a tree structure which is less dependent on the coordinate system so that translations are not as big a problem. The next two chapters explore this idea.

## CHAPTER 4 - THE RECTANGLE-TREE

Three-dimensional tree decompositions have analogous counterparts in two dimensions. For instance, the two-dimensional counterpart of the octree is the quadtree. These two-dimensional versions are easier to visualize and explain; therefore, the discussion of irregular tree decompositions best begins in the two-dimensional realm.

The idea of an irregular decomposition of a two-dimensional region is not new. Pfaltz and Rosenfeld [43] presented the idea of representing a region as a set of maximal neighborhoods in 1967. This idea is a variation on Blum's Medial Axis Transformation [6]. In this representation, only the skeletal points and their respective radii need be stored to fully represent the region. However, since the areas of the blocks centered on the skeleton overlap, there is a redundancy of information resulting in less than optimal storage requirements. Also, though this skeletal representation is better suited to the problem of determining if a point belongs to the region than the boundary representation, the skeletal blocks are organized only as a simple list, which is not the most efficient for queries. The irregular decomposition of interest to this discussion should provide both query speed and space efficiency.

Ferrari, Sankar, and Sklansky [12] have developed an algorithm that finds the minimal rectangular partition of a digitized region. This representation does not have the redundancy of the skeletal representation and may therefore be more space efficient. Still, the partition is not organized in a manner that will result in particularly fast region queries.

The quadtree, on the other hand, provides for efficient queries through its hierarchical structure. Taking advantage of this basic structure, Samet [53] presents a tree version of the Medial Axis Transform which he calls the Quadtree Medial Axis Transform (QMAT). This representation provides space efficiency beyond that of the quadtree and eliminates some of the variability of quadtree size when the region is translated. Unfortunately, since the representation is based on a regular decomposition associated with fixed coordinate axes, the QMAT must still be reconstructed when a translation occurs. This dependence of the representation on position is something that an irregular decomposition should avoid.

#### **Description of the Rectangle-tree**

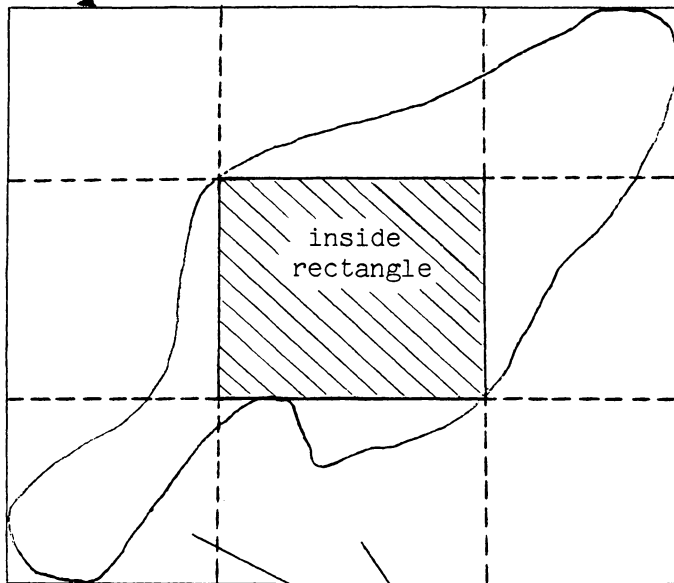
The rectangle-tree provides a representation of a region that is efficient both in space and query speed. The tree is based on an irregular decomposition that is independent of a region's position with respect to translation. Since the

rectangle-tree is composed of disjoint rectangles whose sides are parallel to fixed coordinate axes, the representation is not independent of a region's orientation.

The decomposition used by the rectangle-tree is illustrated in Figure 8. The region is surrounded first by an enclosing box. This box is the smallest rectangle that contains all of the region and has sides parallel to the world coordinate axes. For digital images, the sides will be parallel to the image boundary. After this outside rectangle has been determined, the inside rectangle must be found. The inside rectangle is any largest rectangle that is completely within the region and which has sides parallel to the outside rectangle. The extension of the sides of the inside rectangle divides the outside rectangle into nine, ordered children rectangles. The first child is the inside rectangle, which is known to be completely inside the region. The other eight children are the rectangles derived by extending the sides of the inside rectangle as indicated by the dotted lines in the figure. These children rectangles become the outside rectangles for recursive decomposition. They are ordered according to the size of their inside rectangles, those with the largest inside rectangles being first. Any derived children that contain no part of the region are not included in the tree.

Any rectangle (node) in the tree which is completely contained within the region is labeled "full". Inside rectangles

outside rectangle



derived rectangles

Figure 8. Rectangle-tree Decomposition.

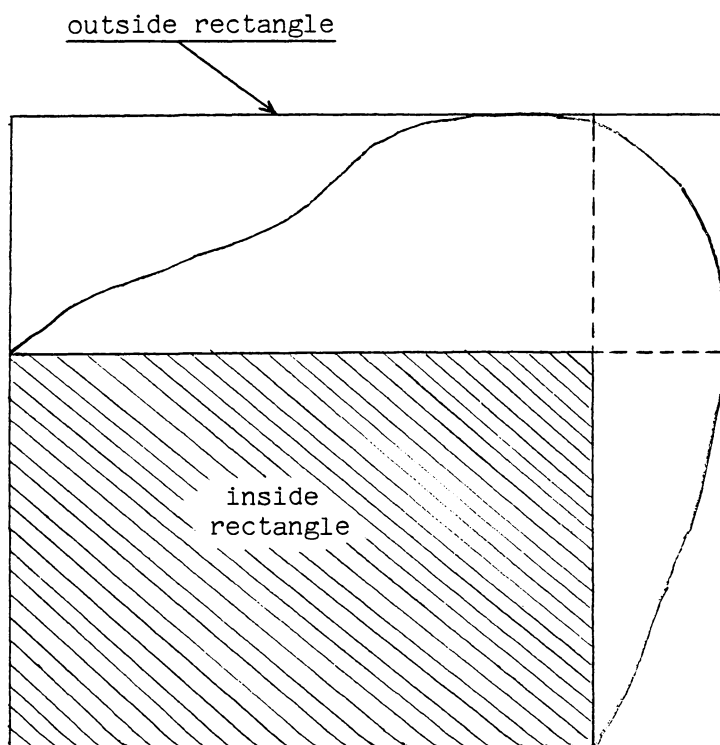


Figure 9. Children at Level 2 Node.

are always full. Full nodes require no further subdivision and are therefore leaves of the tree. Nodes not "full" are "mixed" and are recursively subdivided until either the desired accuracy is achieved or no "mixed" rectangles remain. Figure 9 illustrates the children obtained from the second child of the outside rectangle of Figure 8.

### Experimental Comparison to Quadtree Structures

For the purposes of comparison, rectangle-trees and quadtree-type structures were all stored in a similar format. The trees were stored using a generalized tree structure in a random access file on disk. Each record in the file was a node in the tree.

A generalized tree structure is one in which each node in the tree may have both children and siblings. The children of a node are nodes at the next level of the tree. These nodes represent a decomposition of the parent rectangle for trees such as quadtrees and rectangle-trees. The siblings of a node are nodes at the same level of the tree that have the same parent node. Figure 10 shows the generalized tree structure.

The rectangle-tree record format is depicted in Figure 11. Each record has seven fields. The first four fields contain the first row, last row, first column, and last column of the rectangle which constitutes the tree node. Next is the pointer to the remaining non-empty siblings of the node and

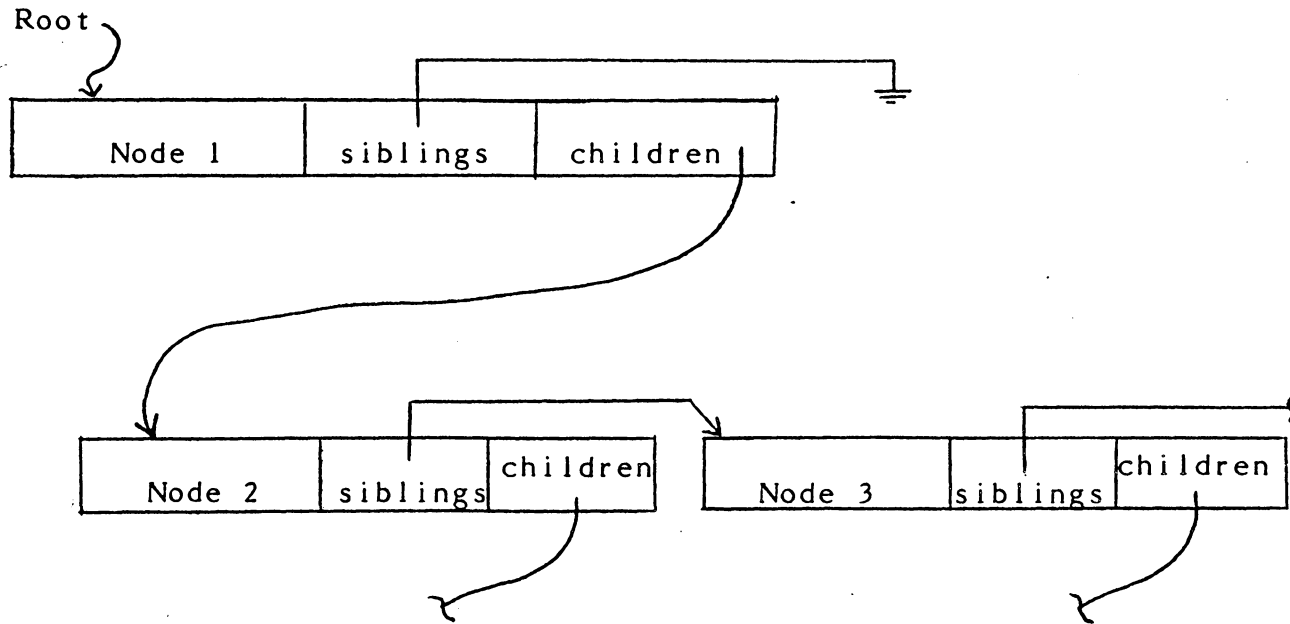


Figure 10. Generalized Tree Structure

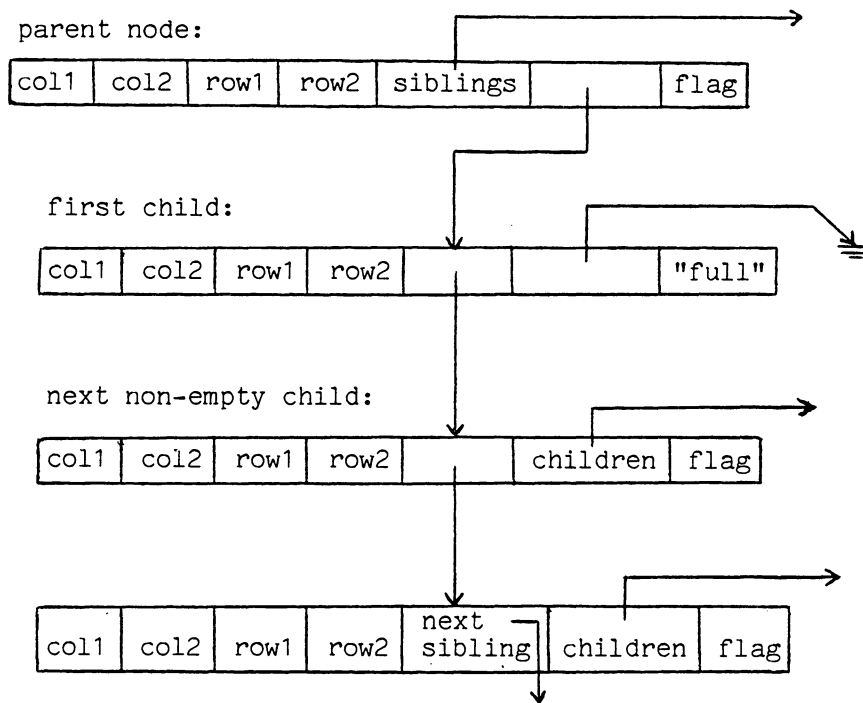


Figure 11. Rectangle-tree Structure.

the pointer to the node's children. If this node is not "full" then the first child will be the node's inside rectangle. The final field of the record is the flag to indicate whether a node is "full" or "mixed".

Similarly, the quadtree data structure has a record format as shown in Figure 12. The principle difference from the rectangle-tree structure is that each node of a quadtree generally has either four children or none. Also, the flag of field seven may take on the values "full", "mixed", or "empty", for a quadtree.

To test the efficiency of the rectangle-tree with respect to queries, several experiments were conducted. For these experiments polygonal regions were randomly generated in square images in the following manner. The origin was assumed to be located at the center of the image. A length between zero and one-half the image size was randomly generated. Then an angle between  $0^\circ$  and  $90^\circ$  was randomly selected. This length and angle determined the location of the first vertex of the polygon in the image. Lengths less than half the image size and angles between  $0^\circ$  and  $121^\circ$  were then randomly generated until the total angle exceeded  $360^\circ$ . The generation of a polygon is illustrated in Figure 13, where the  $l_i$  and  $a_i$  are the generated lengths and angles. The vertices are connected to form the polygon as shown. Once the polygon has been created the region can be colored in to form the test image.

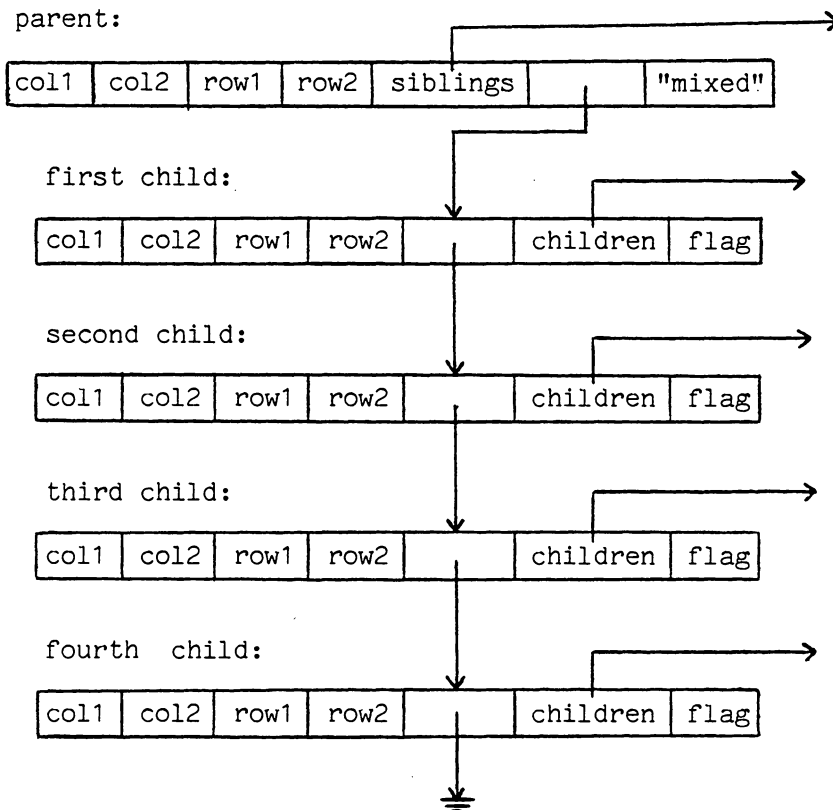


Figure 12. Quadtree Structure.

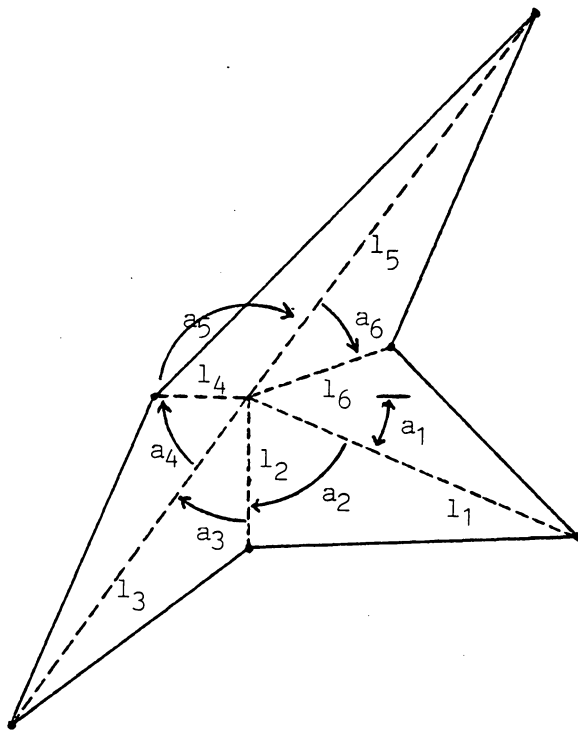


Figure 13. Random Generation of Polygonal Regions.

One query of interest might be to determine if a randomly generated test rectangle intersects the randomly generated polygonal region. The algorithm for this query when the region is represented by a quadtree is given in Figure 14. The algorithm is slightly modified for rectangle-tree representations, as shown in Figure 15. A simple routine INTERSECTS is assumed in these algorithms to determine whether or not two rectangles intersect. Also, a routine READ is assumed that obtains tree node pointed to by X. In the experiments, the number of accesses to the data structure, i.e. the number of calls to READ, was the quantity of interest, since this corresponds to the amount of processing done by each algorithm.

Rectangles for queries were randomly generated by first generating a column value in the image. This was the last column of the rectangle. Then the first column was generated by randomly selecting a column between one and the last column. The last and first rows of the test rectangle were generated in a similar manner.

Table 1 shows the results of two comparison runs for 256×256 images of random polygonal regions. Note that the quadtree representation used in these runs is a complete quadtree where empty nodes are explicit in the tree. These nodes provide no help in determining test rectangle intersections. Therefore, if empty nodes are left out of the

```

procedure QTREE_INT( X, RECTANGLE )
    // intersect RECTANGLE with quadtree pointed to by X //

    if ( X ≠ 0 ) then
        [ call READ( X, RECTANGLE_X, CHILDREN_X,
                    SIBLINGS_X, STATUS )

          if INTERSECTS( RECTANGLE_X, RECTANGLE ) then
              if ( STATUS = full ) then
                  QTREE_INT = true
              else
                  QTREE_INT = QTREE_INT( CHILDREN_X, RECTANGLE )
              if ( QTREE_INT ≠ true ) then
                  QTREE_INT = QTREE_INT( SIBLINGS_X, RECTANGLE ) ]
    else
        QTREE_INT = false

    return

end QTREE_INT

```

Figure 14. Rectangle Intersection Algorithm for Quadtree

```

procedure RTREE_INT( X, RECTANGLE )
    // intersect RECTANGLE with rec-tree pointed to by X //
    if ( X ≠ 0 ) then
        [ call READ( X, RECTANGLE_X, CHILDREN_X,
                    SIBLINGS_X, FULL )

          if INTERSECTS( RECTANGLE_X, RECTANGLE ) then
              if FULL then
                  RTREE_INT = true
              else
                  [ call READ( CHILDREN_X, IN_RECTANGLE,
                              DUMMY, SIBLINGS_CHILD, FULL )

                    if INTERSECTS( IN_RECTANGLE, RECTANGLE ) then
                        RTREE_INT = true
                    else
                        RTREE_INT = RTREE_INT( SIBLINGS_CHILD,
                                                RECTANGLE ) ]

                    if ( RTREE_INT ≠ true ) then
                        RTREE_INT = RTREE_INT( SIBLINGS_X, RECTANGLE )
                ]
            ]
        else
            RTREE_INT = false
    return
end RTREE_INT

```

Figure 15. Rectangle Intersection Algorithm for Rectangle-tree

quadtree structure, giving what shall be called a black quadtree, the results of Table 2 are obtained.

Table 1. Runs on Complete Quadtrees

Run	1	2
Image size	256x256	256x256
No. of images	49	50
No. of test rectangles per image	50	50
Avg. no. of intersections	20.92	23.50
Avg. no. of non-intersections	29.08	26.50
Avg. no. of accesses for quadtree	17.45	17.23
Avg. no. of records in quadtree	1497.57	1597.88
Avg. no. of accesses for rectangle-tree	3.24	3.49
Avg. no. of records in rectangle-tree	325.00	338.78

Note also that the rectangle-tree encodes only the area of the image enclosed by the rectangle which bounds the polygonal region. A similar technique could be employed for quadtrees, i.e. first surround the region with a minimal rectangle, then make regular divisions of this rectangle to

Table 2. Black Quadtree Comparison

Run	3
Image size	256x256
No. of images	49
No. of test rectangles per image	50
Avg. no. of intersections	23.53
Avg. no. of non-intersections	26.47
Avg. no. of accesses for quadtree	11.02
Avg. no. of records in quadtree	975.76
Avg. no. of accesses for rectangle-tree	3.67
Avg. no. of records in rectangle-tree	357.59

form the tree. This should lead to faster determination of null intersections for black quadtrees. The results for this modified black quadtree structure are given in Table 3, and are, in fact, an improvement over the black quadtree. However, since the minimal rectangle enclosing the region is not  $2^n \times 2^n$ , in general, divisions can not be guaranteed to be exactly equal. Thus some of the encoding techniques for quadtrees are not applicable to this modified structure.

Table 3. Modified Black Quadtree Comparison

Run	4	5	6
Image size	500x500	200x200	200x200
No. of images	1	10	46
No. of test rectangles per image	50	20	50
Avg. no. of intersections	27	8.80	25.59
Avg. no. of non-intersections	23	11.20	24.41
Avg. no. of accesses for quadtree	11.16	6.12	7.25
Avg. no. of records in quadtree	no data	817.2	781.1
Avg. no. of accesses for rectangle-tree	6.06	4.29	4.34
Avg. no. of records in rectangle-tree	no data	302.6	289.4

To test the space efficiency of certain tree structures Tamminen [63] uses a  $2^{10} \times 2^{10}$  encoding of a black disk. Following his lead, the  $1024 \times 1024$  black disk was used to compare quadtrees and rectangle-trees. The quadtree of this region was found to have 10,341 nodes, 3,932 of which were black leaves. The rectangle-tree, on the other hand, had 2,354 nodes, 1,117 of which were black leaves. At first glance rectangle-trees would appear to offer a significant advantage, however it should be remembered that the regular-

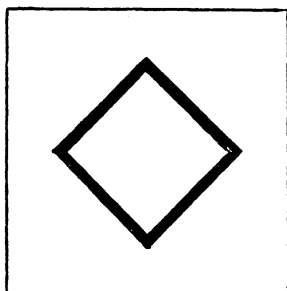
ity of the quadtree decomposition provides certain possibilities for encoding.

Gargantini [16] has show that it is only necessary to store the black leaves of the quadtree and each leaf node can be encoded in  $3(n-1)+2$  bits for a  $2^n \times 2^n$  image. So that for this example, only  $29 \times 3932 = 113,028$  bits are required for the entire quadtree representation.

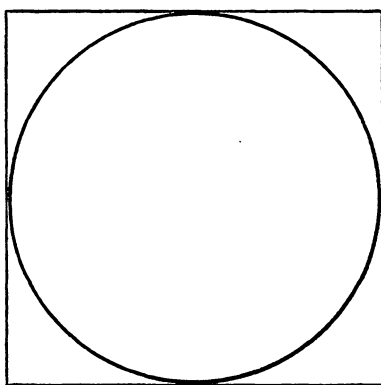
Consider now the rectangle-tree. To represent rectangles in a  $1024 \times 1024$  image, 40 bits are needed, (10 bits for each integer.) Also, 12 bits are needed for each pointer and one additional bit for the flag field. This means that 65 bits are required to encode each node. Recall, however, that the sibling nodes are derived from the outside and inside rectangles of the parent node. Thus if the outside and inside rectangles are completely specified, only a 3-bit code which indicates how to derive the child from these two rectangles needs to be stored in all the nodes which are siblings to the inside node. For rectangle-trees the number of siblings will be one less than the total number of black leaves, since the root node is not a sibling and for each rectangle that is a leaf there is a corresponding outside rectangle which is a sibling. So, of the 3,932 nodes of the example rectangle-tree, 1,176 will be derived rectangles that only require the 3-bit code. Therefore the rectangle-tree can be encoded in  $28 \times 1176 + 65 \times 1178 = 109,498$  bits, still a 3.97% improvement over Gargantini's quadtree encoding.

The reader may have noted that the rectangle-tree should perform well for regions where significant inside rectangles can be found. Consider the images depicted in Figure 16 as cases where the rectangle-tree should do poorly. For the rotated square of Figure 16(a) the quadtree consisted of 1,365 nodes, 348 of which were black leaves, while the rectangle-tree had 489 nodes and 245 black leaves. For the circle encoded as in Figure 16(b), the quadtree had 15,189 nodes, 3,244 black leaves, and the rectangle-tree had 2,391 nodes with 1,307 black leaves.

The performance of both types of tree structures is highly data dependent. Obviously, the rectangle-tree will perform miserably on a pixel-sized checker board. But, since the desired use for this decomposition is for representing real objects (that generally have some substantial area) and not just arbitrary images, the rectangle-tree's poor performance on the checker board is not of particular concern. However, the practicality of this type of decomposition in three-dimensions should be examined in some detail. Enter Chapter 5.



(a) 128 x 128



(b) 1024 x 1024

Figure 16. Line Images.

## CHAPTER 5 - THE RECTANGULAR PARALLELEPIPED-TREE

The extension of the rectangle-tree to three-dimensional shape representation yields a tree of rectangular parallelepipeds (RPP's). Thus this decomposition can be termed a "rectangular parallelepiped-tree" or "RPP-tree". Sometimes the term "R-tree" will be used to refer to both rectangle- and RPP-trees.

Figure 17 depicts the three-dimensional decomposition of RPP-trees. This decomposition is completely analogous to the rectangle-tree structure of two dimensions. All of the RPP's in the RPP-tree have edges parallel to the coordinate axes of a world coordinate system. The RPP-tree has an inside RPP and an outside RPP at each node which is "mixed". The planes of the boundary of the inside RPP partition the outside RPP into 27 smaller RPP's. These 27 RPP's, which include the inside RPP, form the children of the outside RPP. Any "mixed" children are recursively subdivided in a similar manner to form the entire tree.

Since the increase in dimensionality has led to a significant increase in the number of possible children per node, the performance of the RPP-tree should be examined closely.

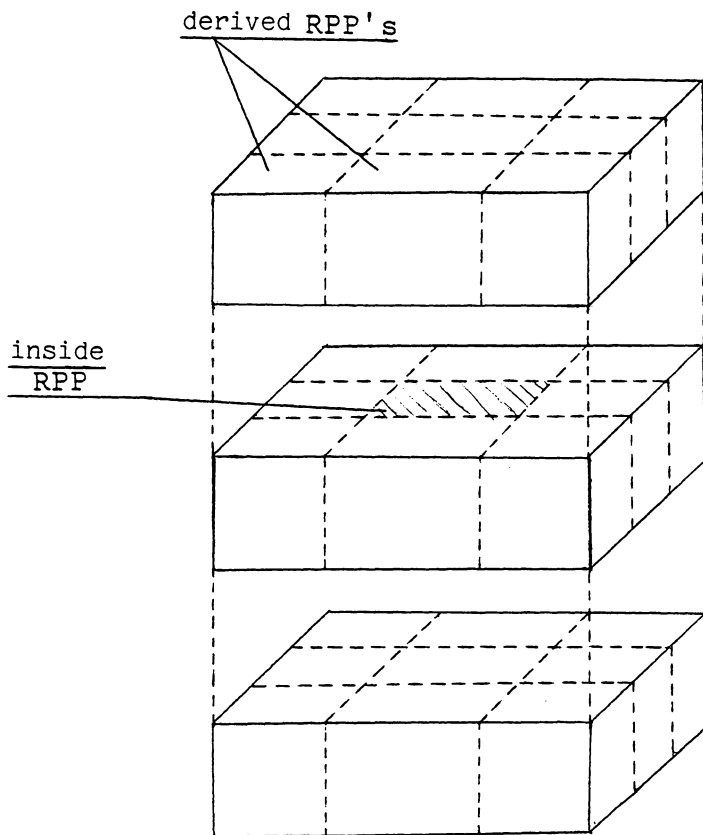


Figure 17. RPP-tree Decomposition

## Experimental Comparison to Octree Structures

In GIPSY, voxel arrays are stored as multi-band images. These multi-band images were used to conduct experiments on the properties of RPP-tree and octree encodings of three-dimensional objects.

The RPP-tree was stored in the same manner as the rectangle-tree of Figure 11. Two fields were added to each record to indicate the first and last bands of the record's RPP. The other fields, including the pointer and flag fields, retained their original definitions. Likewise, octrees were stored as in Figure 12 with the addition of fields indicating the first and last bands of each RPP.

For the four-sided polyhedron of Figure 18, positioned in a  $64 \times 64 \times 64$  voxel array as shown, the following results were obtained. The octree contained 12,089 nodes, 4,682 of which were black leaves. The RPP-tree contained 2,852 nodes, 1,434 of which were black leaves.

The intersection algorithms of Figures 14 and 15 can be transferred directly to the three-dimensional domain by simply replacing all references to rectangles with rectangular parallelepipeds. Because the algorithms are based on a generalized tree structure, no other changes are required. These algorithms were used to compare the efficiencies of RPP-trees and octrees with respect to queries. For 50 randomly generated RPP's within the array of Figure 18, the RPP-tree had

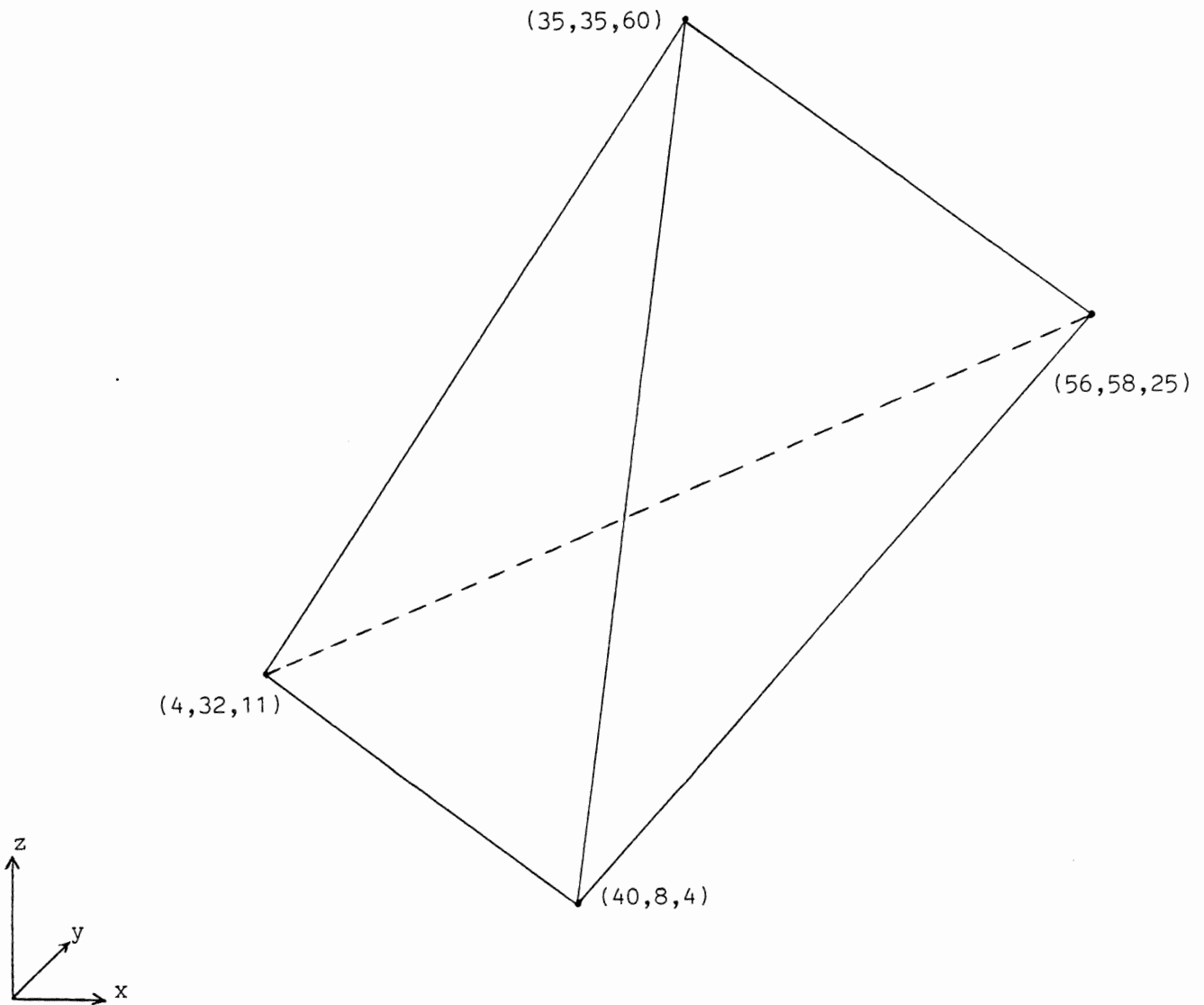


Figure 18. Four-sided Polyhedron

an average data structure access count of 22.64 accesses per rectangle, while the octree had an average access count of 34.54. However, for the black octree, i.e. the tree containing only mixed and black nodes, the speed of the octree query was much improved. For 50 random RPP's, the RPP-tree averaged 24.46 accesses and the black octree, only 22.66 accesses.

For objects that have a little more regularity with respect to the coordinate axes, the properties of the RPP-tree become a distinct advantage. Consider the quadrilateral cylinder positioned in a  $64 \times 64 \times 64$  voxel array as shown in Figure 19. The octree for this object had 22,569 nodes, while the RPP-tree required only 137 nodes, a significant difference to say the least. The black octree for this example contained 12,148 nodes. The average number of accesses for the black octree was 29.24 compared with 4.76 for the RPP-tree, when 50 random intersection RPP's were tested.

Due to the irregular nature of the RPP-tree decomposition, its performance is not affected by a change in image resolution, whereas the octree's performance degrades as the resolution is increased. For example, for quadrilateral cylinders, the octree does much better against the RPP-tree at a coarser resolution. In Figure 20, a quadrilateral cylinder similar to that of Figure 19 is positioned in a  $25 \times 25 \times 25$  voxel array. The octree-like decomposition of this volume yielded a tree of 1607 nodes, while the RPP-tree had 52 nodes. Also, for determining the intersection of 20 randomly

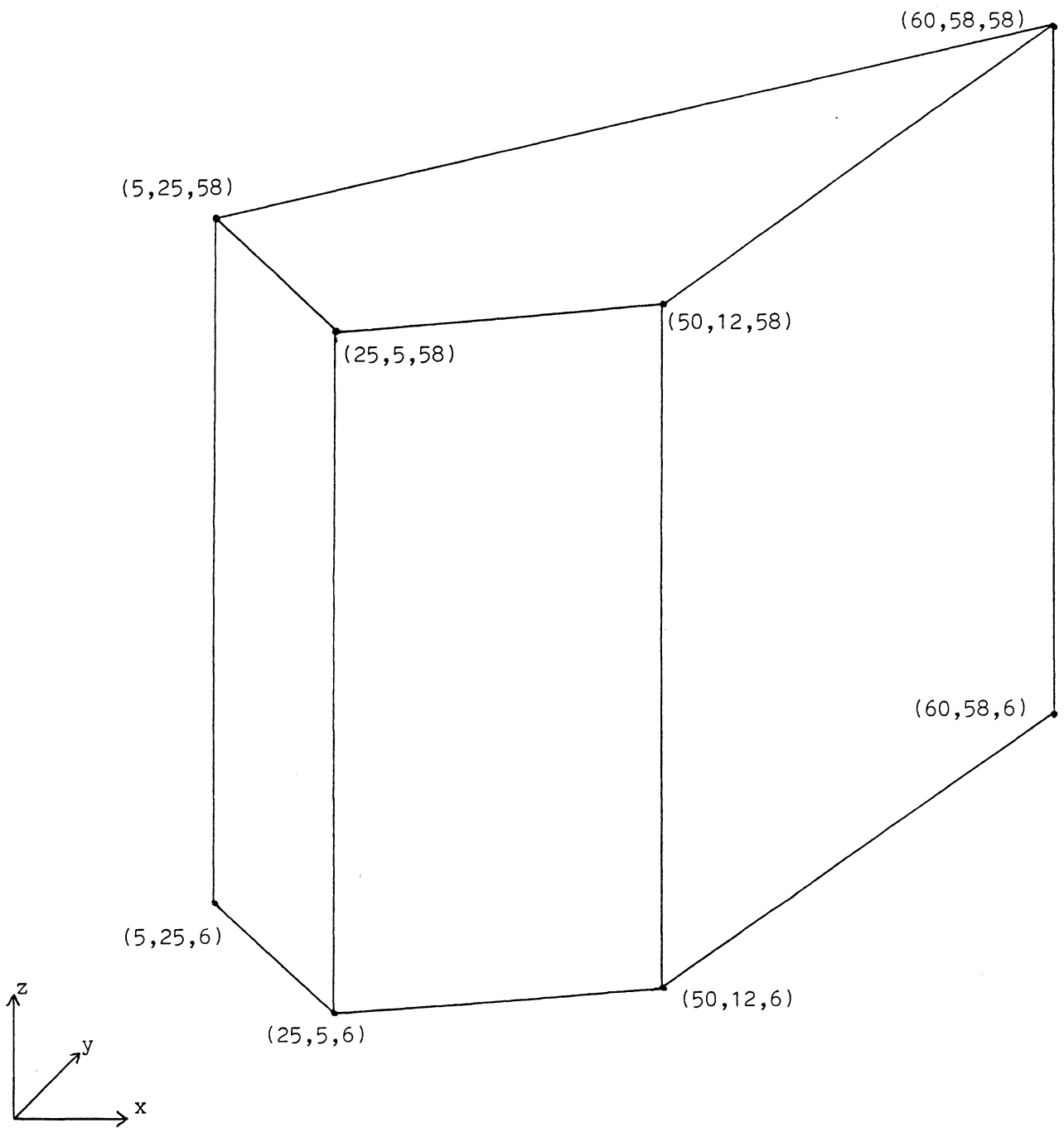


Figure 19. Quadrilateral Cylinder

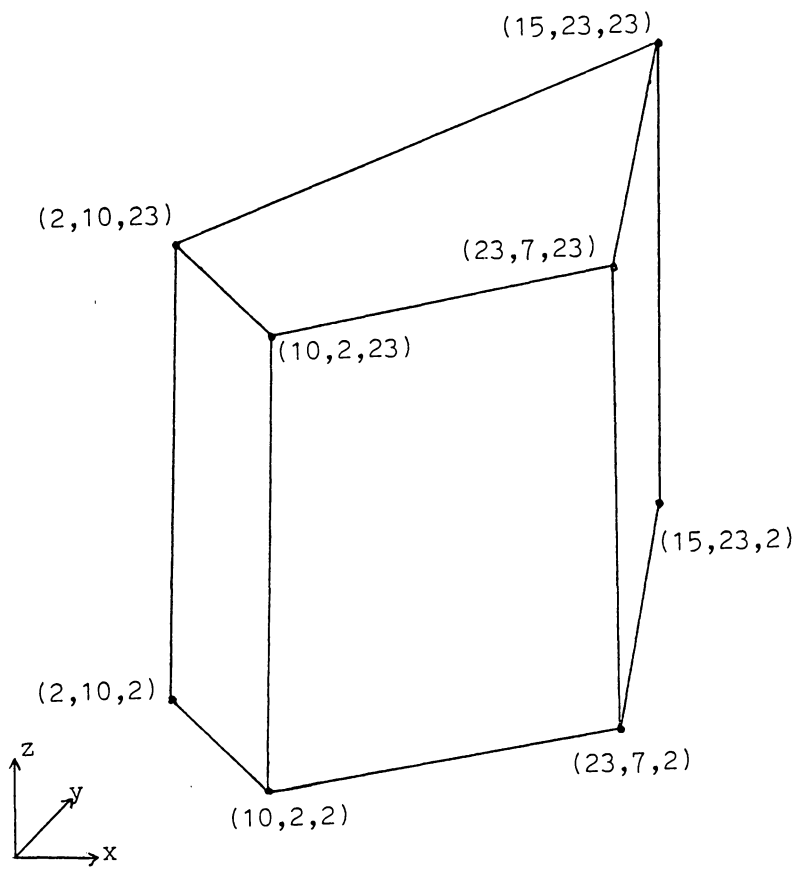


Figure 20. Quadrilateral Cylinder at Corser Resolution

generated RPP's with the object, the octree structure required 11.8 accesses and the RPP-tree, 3.9 accesses, on the average.

Since the size of the octree is highly dependent on the position of the world coordinate origin, it may be possible to obtain a significant improvement by selecting a more appropriate origin when constructing the tree. To test this idea, the origin in Figure 20 was moved to the point where the inside RPP of the root of the RPP-tree would correspond to an entire octant of the octree. This experiment led to an octree of 943 nodes, a significant improvement over the 1607 nodes of the previous tree. However, the RPP-tree still only contains 52 nodes. It is not affected by translation!

As a final experiment, the "worst case" of RPP-tree space efficiency was tested. In Chapter 4, the worst case space efficiency turned out to be a thin square with no sides parallel to the coordinate axes. (Figure 16(a).) For the three-dimensional worst case consider the cube of Figure 21 rotated by  $45^\circ$  around the x-, y-, and z-axes, and translated 32 units in the positive x-, y-, and z-directions into the center of a  $64 \times 64 \times 64$  image. The object that was encoded was the one-pixel-wide solid that comprised the surfaces of the cube. The octree for this object contained 8,145 nodes, 2,355 of which were black leaves and 4,772 of which were white leaves. The RPP-tree had 2,281 nodes, 1,169 of which were black leaves.

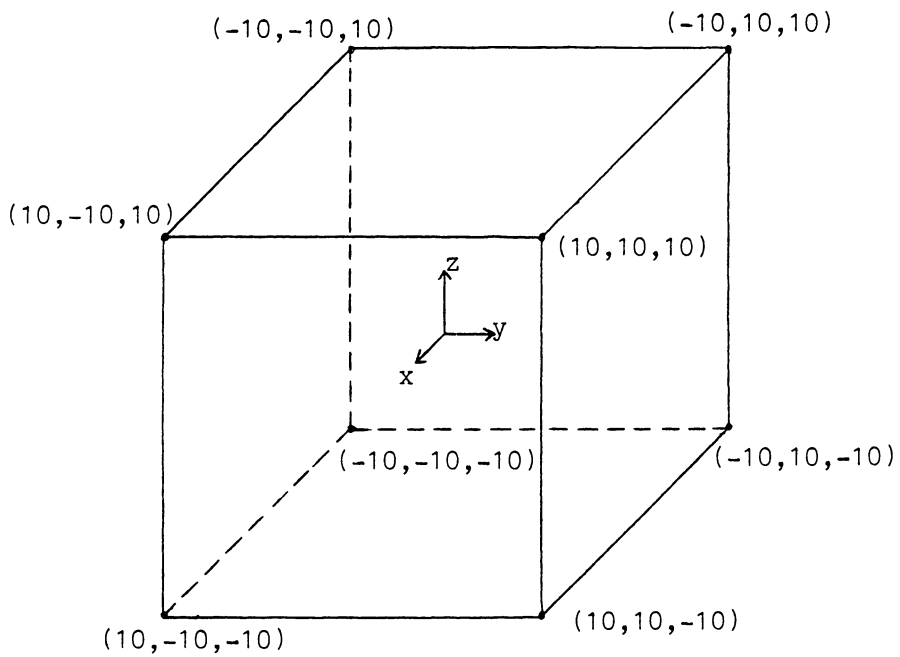


Figure 21. Cubic Shell

Again, as in the two-dimensional case, the performance of the R-tree is highly data dependent. But for a machined part, such as an aircraft bulkhead, described as a set of parts with fillets removed, there may be enough regularity of shape to give the RPP-tree a distinct advantage. The RPP-tree also has the advantage of being invariant under transformations, which is particularly useful in the inspection scenario. The problem remains, however, of obtaining tree decompositions from the relational model of the object. This problem is addressed in the next chapter.

## CHAPTER 6 - OBTAINING R-TREES

As mentioned previously tree decompositions, such as octrees and R-trees, provide advantages over boundary and CSG representations for problems related to robot planning. Obviously, entering these trees by hand would be completely intractable. Therefore algorithms to obtain them from other representations are needed. This chapter will examine, in particular, the problem of obtaining R-trees from the relational model representation of Chapter 3. However, many of the results will be more generally applicable.

The problem of constructing an R-tree of a part is mainly one of determining the inside and outside RPP's that make up the tree. The outside RPP is the minimal, enclosing RPP of the part, that has edges parallel to the world coordinate axes. The inside RPP is the near maximal RPP that has edges parallel to those of the outside RPP, and is completely enclosed by the part. The problem of finding these two types of RPP's spawns two subproblems, one of determining if a point on a surface is within a boundary, and another of finding a discretized representation of a part.

This chapter will present solutions to all of these problems, but in order to facilitate the discussion, the notation to be used will first be formalized.

## Notation

In this discussion, upper case will be used to represent matrices, bold lower case will represent vectors, and lower case will signify scalars.

The representation of Chapter 3 and the development which follows both involve homogeneous coordinate systems. A space vector in homogeneous coordinates is written

$$\mathbf{v} = \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} .$$

The true  $x$ ,  $y$ , and  $z$  coordinates of the vector  $\mathbf{v}$  are  $x/w$ ,  $y/w$ , and  $z/w$ ; however  $w$  is usually taken to be unity to simplify things. The principle advantage of homogeneous coordinates is that a rigid body transformation involving both rotation and translation, can be written as a single matrix,

$$T = \begin{bmatrix} o_1 & n_1 & p_1 & t_1 \\ o_2 & n_2 & p_2 & t_2 \\ o_3 & n_3 & p_3 & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} .$$

The vectors  $o$ ,  $n$ ,  $p$  are orthonormal and represent the rotational part of the transformation. The vector  $t$  represents the translational part.

Note that the inverse of  $T$  can easily be computed as

$$T^{-1} = \begin{bmatrix} o_1 & o_2 & o_3 & -o \cdot t \\ n_1 & n_2 & n_3 & -n \cdot t \\ p_1 & p_2 & p_3 & -p \cdot t \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

As mentioned in Chapter 3, in the relational model, lower level coordinate systems are related to higher level coordinate systems by the use of these homogeneous coordinate transformations. These coordinate systems are called frames and include Frame(world), Frame(object), Frame(part), Frame(surface), Frame(piece) and Frame(arc). Recall that Frame(arc) is referred to as the "lowest level" of the model. See Figure 7. A transformation that relates a pair of these frames can be given two interpretations. One, the transformation represents a change of coordinates in going from a higher level frame to a lower level frame. Or, alternatively, the transformation transforms the points of the entities at a lower level into the points of the entity represented at a higher level.

For example, suppose a quadric equation in Frame(piece) is given by

$$\begin{aligned}
 ax^2 + 2bxy + cy^2 + 2dyz + ez^2 + 2fxz \\
 + 2gx + 2hy + 2iz + j = 0. \quad (3)
 \end{aligned}$$

Suppose further that this piece is a part of a surface described in Frame(surface) and that the transformation that relates Frame(surface) and Frame(piece) is given by T.

Equation (3) can be represented in vector form by

$$v^t P v = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} a & b & f & g \\ b & c & d & h \\ f & d & e & i \\ g & h & i & j \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = 0.$$

Any quadric can be so represented in a quadratic form involving a real, symmetric matrix P [62]. It will be assumed here that all matrices that describe quadrics are symmetric.

Now, if an equation for this quadric is desired in the higher (surface) coordinate frame, a transformation of variables is required such that the coefficients for this piece equation referred to Frame(surface) are obtained. Thus, a transformation that changes the coordinates of the lower level piece system to that of the higher level surface system is desired. This action is the opposite of that of T as defined in the relational model. Therefore, the transformation needed is  $T^{-1}$  instead of T.

The equation in Frame(surface) is thus

$$(T^{-1}v)^t P (T^{-1}v) = v^t (T^{-1})^t P (T^{-1})v = v^t Q v = 0, \quad (4)$$

where

$$Q = (T^{-1})^t P (T^{-1})$$

is the coefficient matrix for the quadric expressed in Frame(surface) [13]. Note that if  $P$  is symmetric then  $Q$  is also symmetric. This is an example of one of the ways in which  $T$  is used.

Now consider, if a point  $p = (p_x, p_y, p_z, 1)^t$  in Frame(piece) satisfies (3), the corresponding point  $q$  in Frame(surface) is given by

$$q = Tp.$$

This is verified by,

$$p^t P p = 0,$$

and

$$q^t Q q = (Tp)^t Q (Tp) = p^t T^t (T^{-1})^t P (T^{-1}) Tp = p^t P p = 0.$$

This is an example of the other type of interpretation that can be given to the transformation that relates two frames of the relational model.

### The Outside RPP of a Part

The outside rectangular parallelepiped (RPP) of a solid may be determined by locating the solid's extrema in the coordinate directions of the world coordinate system.

A solid in the relational model of Chapter 3 is represented as a part. A part is represented by surfaces and edges. Each surface is composed of a number of surface patches called pieces. And a piece may be described in either of two ways, as a quadric surface equation,

$$f_p = v^t P v = 0, \quad (5)$$

or as a linear combination of parametric basis functions,

$$f(u,v) = CB, \quad (6)$$

where  $C = 4 \times n$  coefficient matrix,

$B = n \times 1$  basis function vector,

$u, v =$  parametric variables.

In the second case,  $f$  is a vector function of  $u$  and  $v$ . The surface patch this function represents is thus the graph of this function in  $R^3$ . The form of (5) has been called an implicit representation [56], and may be thought of as the graph of parametric vector functions. The parametric functions for which (5) is the graph will be referred to here as the implicit functions of (5). Although (5) can have implicit functions  $y = h(x, z)$  or  $x = k(y, z)$ , the form  $z = g(x, y)$  will be used in this development for definiteness.

The implicit function theorem of vector calculus, see [37], states the conditions under which an implicit function can be found. If

$$f_p(v_0) = 0 \quad \text{and} \quad \frac{\partial f_p(v_0)}{\partial z} \neq 0,$$

then  $z$  can be expressed as  $z = g(x, y)$  for a small ball surrounding  $v_0$ . Furthermore,

$$\partial g / \partial x = - \partial f_p / \partial x \div \partial f_p / \partial z,$$

$$\partial g / \partial y = - \partial f_p / \partial y \div \partial f_p / \partial z.$$

This implies that (3) can be expressed as a vector function,

$$f_p(x, y) = \begin{bmatrix} x \\ y \\ g(x, y) \\ 1 \end{bmatrix} .$$

Therefore, for either representation of a piece, a vector function of the form,

$$f_p(u, v) = \begin{bmatrix} x(u, v) \\ y(u, v) \\ z(u, v) \\ 1 \end{bmatrix} ,$$

can be obtained. The graph of this function is the surface patch in Frame(piece). This patch in Frame(world) is given by the graph of

$$f_w(u, v) = T \cdot f_p(u, v) ,$$

where T is the transformation relating Frame(piece) and Frame(world). To locate the critical points (possible extrema) of this function in the x, y, and z directions, differentiate and set the result to zero. This gives

$$\mathbf{0} = \frac{\partial}{\partial u} f_w(u, v) = \mathbf{T} \cdot \frac{\partial}{\partial u} f_p(u, v) =$$

$$\begin{bmatrix} o_1 & n_1 & p_1 & t_1 \\ o_2 & n_2 & p_2 & t_2 \\ o_3 & n_3 & p_3 & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\partial}{\partial u} x(u, v) \\ \frac{\partial}{\partial u} y(u, v) \\ \frac{\partial}{\partial u} z(u, v) \\ 0 \end{bmatrix} =$$

$$\begin{bmatrix} o_1 \frac{\partial}{\partial u} x(u, v) + n_1 \frac{\partial}{\partial u} y(u, v) + p_1 \frac{\partial}{\partial u} z(u, v) \\ o_2 \frac{\partial}{\partial u} x(u, v) + n_2 \frac{\partial}{\partial u} y(u, v) + p_2 \frac{\partial}{\partial u} z(u, v) \\ o_3 \frac{\partial}{\partial u} x(u, v) + n_3 \frac{\partial}{\partial u} y(u, v) + p_3 \frac{\partial}{\partial u} z(u, v) \\ 0 \end{bmatrix} \quad (7)$$

and

$$\mathbf{0} = \frac{\partial}{\partial v} f_w(u, v) = \mathbf{T} \cdot \frac{\partial}{\partial v} f_p(u, v) =$$

$$\begin{bmatrix} o_1 & n_1 & p_1 & t_1 \\ o_2 & n_2 & p_2 & t_2 \\ o_3 & n_3 & p_3 & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\partial}{\partial v} x(u, v) \\ \frac{\partial}{\partial v} y(u, v) \\ \frac{\partial}{\partial v} z(u, v) \\ 0 \end{bmatrix} =$$

$$\begin{bmatrix} o_1 \frac{\partial}{\partial v} x(u, v) + n_1 \frac{\partial}{\partial v} y(u, v) + p_1 \frac{\partial}{\partial v} z(u, v) \\ o_2 \frac{\partial}{\partial v} x(u, v) + n_2 \frac{\partial}{\partial v} y(u, v) + p_2 \frac{\partial}{\partial v} z(u, v) \\ o_3 \frac{\partial}{\partial v} x(u, v) + n_3 \frac{\partial}{\partial v} y(u, v) + p_3 \frac{\partial}{\partial v} z(u, v) \\ 0 \end{bmatrix} \quad (8)$$

Thus for each direction, x, y, and z, there are two equations in u and v to be solved for all possible extremal points  $(u_0, v_0)$ . Each of these points may be tested as an extrema by plugging in the appropriate Hessian matrix. For example, the Hessian for the x-direction is given by,

$$H_x = \begin{bmatrix} \left[ \begin{array}{c} \frac{\partial^2}{\partial u^2} f_w \\ \frac{\partial^2}{\partial u \partial v} f_w \end{array} \right]_x & \left[ \begin{array}{c} \frac{\partial^2}{\partial u \partial v} f_w \\ \frac{\partial^2}{\partial v^2} f_w \end{array} \right]_x \\ \left[ \begin{array}{c} \frac{\partial^2}{\partial v \partial u} f_w \\ \frac{\partial^2}{\partial v^2} f_w \end{array} \right]_x & \left[ \begin{array}{c} \frac{\partial^2}{\partial v^2} f_w \\ \frac{\partial^2}{\partial v^2} f_w \end{array} \right]_x \end{bmatrix}$$

where the x subscript on the inner vectors indicates that only the x-component of the vector is used. If  $H_x$  is positive (negative) definite at  $(u_0, v_0)$ , then  $(u_0, v_0)$  is a local minimum (maximum) in the x-direction.

The primary description case is of special interest, so it will be examined in more detail. The primary description is given by (5) where

$$P = \begin{bmatrix} a & 0 & 0 & d/2 \\ 0 & b & 0 & e/2 \\ 0 & 0 & c & f/2 \\ d/2 & e/2 & f/2 & g \end{bmatrix}$$

Thus

$$\frac{\partial}{\partial \mathbf{v}} [ \mathbf{f}_p(\mathbf{v}) ] = \frac{\partial}{\partial \mathbf{v}} [ \mathbf{v}^t \mathbf{P} \mathbf{v} ] = 2\mathbf{P}\mathbf{v} = \begin{bmatrix} 2ax + d \\ 2by + e \\ 2cz + f \\ 0 \end{bmatrix} .$$

Recall that  $z = g(x, y)$ , so

$$\frac{\partial g}{\partial x} = \frac{-(2ax + d)}{(2cz + f)} ,$$

$$\frac{\partial g}{\partial y} = \frac{-(2by + e)}{(2cz + f)} .$$

Therefore,

$$\frac{\partial}{\partial x} \mathbf{f}_w = \mathbf{T} \cdot \frac{\partial}{\partial x} \mathbf{f}_p = \mathbf{T} \cdot \frac{\partial}{\partial x} \begin{bmatrix} x \\ y \\ g(x, y) \\ 1 \end{bmatrix} = \mathbf{T} \cdot \begin{bmatrix} 1 \\ 0 \\ \frac{\partial g}{\partial x} \\ 0 \end{bmatrix} = \mathbf{T} \cdot \begin{bmatrix} 1 \\ 0 \\ \frac{-(2ax+d)}{(2cz+f)} \\ 0 \end{bmatrix}$$

and

$$\frac{\partial}{\partial y} f_w = T \cdot \frac{\partial}{\partial y} f_p = T \cdot \frac{\partial}{\partial y} \begin{bmatrix} x \\ y \\ g(x, y) \\ 1 \end{bmatrix} = T \cdot \begin{bmatrix} 0 \\ 1 \\ \frac{\partial g}{\partial y} \\ 0 \end{bmatrix} = T \cdot \begin{bmatrix} 0 \\ 1 \\ \frac{-(2by+e)}{(2cz+f)} \\ 0 \end{bmatrix}$$

Thus, for the primary description, (7) and (8) become

$$\begin{bmatrix} o_1 - p_1 \frac{(2ax+d)}{(2cz+f)} \\ o_2 - p_2 \frac{(2ax+d)}{(2cz+f)} \\ o_3 - p_3 \frac{(2ax+d)}{(2cz+f)} \\ 0 \end{bmatrix} = 0 \quad (9)$$

and

$$\begin{bmatrix} n_1 - p_1 \begin{matrix} (2by+e) \\ (2cz+f) \end{matrix} \\ n_2 - p_2 \begin{matrix} (2by+e) \\ (2cz+f) \end{matrix} \\ n_3 - p_3 \begin{matrix} (2by+e) \\ (2cz+f) \end{matrix} \\ 0 \end{bmatrix} = 0 \quad (10)$$

Note that these equations are meaningful only for the directions in which the  $p$  vector is not zero. For instance, if  $p_1 = 0$ , then the  $x$ -direction equations digress to  $o_1 = 0$  and  $n_1 = 0$ , which are of no value in determining extrema. Fortunately, the form of (9) and (10) is a direct result of the choice of the implicit function. Therefore, when  $p_1 = 0$ , one of the other forms,  $y = h(x, z)$  or  $x = k(y, z)$ , could be utilized in obtaining a solution. Since  $o$ ,  $p$ , and  $n$  are orthonormal, one of the implicit forms will provide a valid approach for a given direction. Also, it is not necessary that each direction use the same implicit form in a solution - different implicit functions can be used for the solution of extrema in different directions for a given piece.

Obviously, a solution  $(x_0, y_0, z_0)$  of (9) and (10) must satisfy

$$\frac{\partial}{\partial z} f_p \left( \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{bmatrix} \right) \neq 0.$$

in order to qualify as a possible extrema. If this condition is not satisfied, a different implicit form must be used.

Once a minimum (maximum) has been found, it must be ascertained whether or not this value  $(u_0, v_0)$  occurs in the actual domain of the piece. For the secondary description this is simply a matter of determining if  $(u_0, v_0)$  is within the  $u$ - and  $v$ -intervals specified in the attribute-value table of the secondary description since the parameterization is the same in either coordinate system, i.e.

$$q_w(u, v) = T \cdot q_p(u, v)$$

for all  $u$  and  $v$ .

For the primary description, this is not so simple. The quadric equation describes an entire surface, such as an ellipse, but the piece represented by the description may be only one small patch of this ellipse. The  $x$ ,  $y$ , and  $z$ , values range over the intervals given in the attribute-value table only if the piece is a rectangular one. Otherwise, the values give only a bound on the domain and the actual domain is determined by the boundary of the piece.

The usual technique for describing quadrics in a boundary representation is to use constraining equations on the  $x$ ,  $y$ , and  $z$  parameters [36,68], rather than this more general approach of a boundary of arcs. Constraining equation approaches simplify the solution to the point-membership problem but make the solution of the bounding RPP problem more difficult [68]. The problem of determining if a point satisfying the primary description equation is in the domain of the piece, dubbed the point-in-boundary problem, will be considered in the next section. For now it will be assumed that this problem can be solved so that the solution to the bounding RPP problem can be completed.

If  $(u_0, v_0)$  is in the piece domain, then it could be a global minimum (maximum) for the direction for which is was computed ( $x$ ,  $y$ , or  $z$ .) Thus, compare it to the global minimum (maximum) so far. If it is smaller (larger) then let it be the new global minimum (maximum). If this is done for all pieces of the part, the bounding RPP can be determined as  $(x_{\min}, x_{\max}, y_{\min}, y_{\max}, z_{\min}, z_{\max})$ . However, if  $(u_0, v_0)$  is not in the piece domain, or there are no critical points for a piece, then the arcs of the piece must be checked to see if the minimum (maximum) occurs there.

The problem of finding extrema on arcs is exactly analogous to the surface patch case. The problem is merely one of one fewer dimensions. Thus for the primary description

$$f_a = u^t R u = \begin{bmatrix} x & y & 0 & 1 \end{bmatrix} \begin{bmatrix} a & 0 & 0 & c \\ 0 & b & 0 & d \\ 0 & 0 & 0 & 0 \\ c & d & 0 & f \end{bmatrix} \begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix} = 0,$$

which can, under conditions similar to before, be written as a vector function,

$$f_a(x) = \begin{bmatrix} x \\ g(x) \\ 0 \\ 1 \end{bmatrix}$$

or

$$f_a(y) = \begin{bmatrix} h(y) \\ y \\ 0 \\ 1 \end{bmatrix}.$$

For the secondary description,

$$f_a(t) = CB = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \\ 1 \end{bmatrix},$$

so that the arc can always be represented as a graph of a vector function of one parametric variable.

The critical points, then, are solutions of

$$\frac{\partial}{\partial t} f_w(t) = T \cdot \frac{\partial}{\partial t} f_a(t) = \mathbf{0},$$

exactly like before. The entire solution proceeds as that of the surface case.

One simplification that can be made in the solution to the arc problem is that of not computing the Hessian for arc critical points. Determining whether the Hessian is positive or negative definite will require at least three floating-point comparisons, but it is relatively easy to determine if a point is in the domain of an arc. Thus, it is generally more efficient to locate extrema by comparing all critical points in an arc's domain to the global extrema computed so far, than to compute the Hessian to determine if a critical point is a local minimum or maximum.

Using the results of this section, the algorithm for determining the outside RPP is straight-forward. All pieces are tested for local extrema and these compared to determine the global values. If a piece does not have a local maximum or a local minimum, the arcs of the piece boundary are tested for global extrema. This algorithm results in the minimum  $x$ ,  $y$ , and  $z$  coordinates, and the maximum  $x$ ,  $y$ , and  $z$ , coordinates for the part. These six values define the minimal volume RPP

bounding the part with edges parallel to the world coordinate axes.

### Point-in-Boundary Problem

The problem of determining whether or not a point  $q_0$ , satisfying (5), is inside the area of the surface enclosed by the piece boundary, is closely related to the well-known point-in-polygon problem detailed in [9,20,50,61]. In fact, the point-in-boundary problem can be considered a generalization of the point-in-polygon problem with the restrictions to planar surfaces and boundaries of only straight line segments removed.

The basic idea behind the solution to the point-in-polygon problem is as follows. The boundary of the polygon is directed, just as boundaries of the relational model are directed in a right-handed sense. This means that the segments of the boundary are ordered in a manner that gives one complete traversal of the boundary. Also, each segment is directed from its predecessor to its successor. The boundaries of the relational model are directed in a right-handed sense. Now, if a ray is extended from the point being tested, as shown in Figure 22, it may have a number of intersections with the boundary. Each intersection has a sense, positive or negative, determined by the direction of the cross-product of the segment direction with the ray vector. For example, cross-

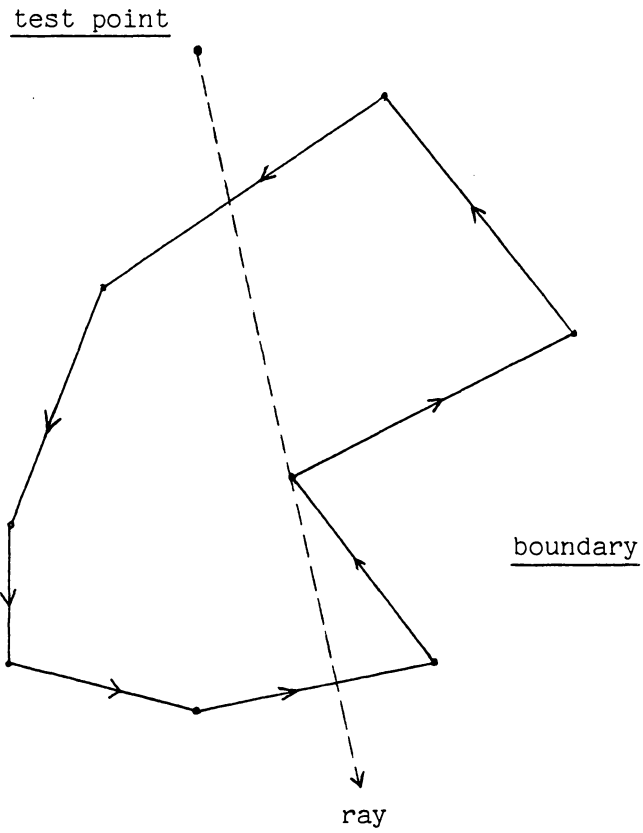


Figure 22. Point-in-Polygon Solution.

products pointing into the page of Figure 22 could be defined to be negative and thus, cross-products pointing out of the page would be positive.

Intuitively, if the number of intersections of the ray with the boundary is even, the point is outside the polygon, but if the number is odd, the point is inside. This is true except when the ray intersects a vertex of the polygon. In this case, the senses of the intersections must be examined to determine whether or not the ray has actually crossed the boundary. If both intersections have the same sense, then the boundary is crossed, otherwise it is not. If a segment of the boundary coincides with the ray, then intersections with that segment are ignored and the intersections with the arcs at the segment's endpoints are treated together as a vertex intersection. Thus, for straight line boundaries, the number of crossings can always be determined and the point-in-polygon query correctly answered.

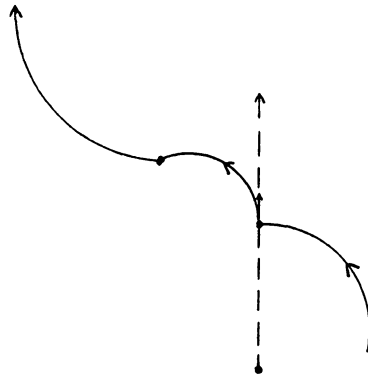
However, if this particular technique is generalized to handle boundaries of planar surfaces which include non-linear arcs, a difficulty arises. For non-linear arcs, the natural vector to represent the direction of the arc would be a vector lying along the tangent to the arc at the point of intersection. This works fine except in the case of Figure 23. Here, the ray intersects a vertex of the boundary in such a way as to coincide with the direction of the tangent of arc<sub>j</sub>. Thus, there is no way to distinguish the boundary-

crossing intersection of Figure 23(a) from the non-crossing intersection of Figure 23(b), and the point-in-boundary query can not be answered. Due to this complication, the basic idea for the point-in-boundary query must be slightly modified from that of the point-in-polygon query.

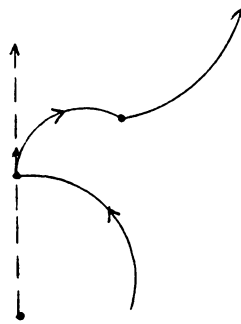
The solution to the point-in-boundary problem is based on the idea that, for any point on the surface and any simply-connected boundary of arcs also on the surface, a line can be drawn along the surface from the point to some arc of the boundary without crossing any other arcs of the boundary. Figure 24(a) shows this type line from point to boundary, while Figure 24(b) depicts an invalid line connecting point and boundary. Once such a line is created, the right-handedness of the boundary, the direction from point to boundary along this line, and the direction of the outward surface normal, can all be used to determine if the point is inside or outside the boundary.

The complete algorithm for determining if a point  $q_0$  lies inside the piece boundary is as follows. (It is assumed that the point does not lie on the boundary itself.)

1. Choose an arc from the list of arcs given in the piece boundary. Call this arc  $j$ .
2. Find a point  $q_1$  on arc  $j$  which is not an endpoint of arc  $j$ .



(a) boundary crossing



(b) non-crossing boundary intersection

Figure 23. Indistinguishable Boundary Intersections.

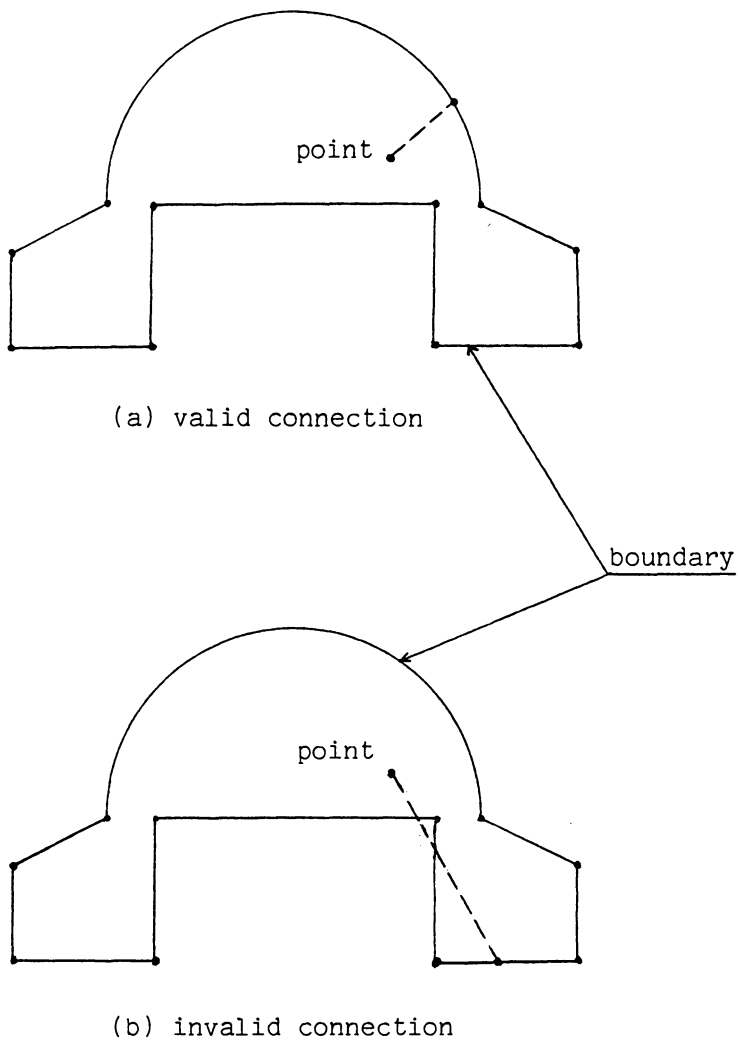


Figure 24. Point-to-Boundary Connections.

3. Calculate the surface normal at  $q_1$ . By the relational model conventions previously described, the outward surface normal is

$$n(q_1) = \frac{\partial}{\partial v} (v^t P v) |_{q_1} = 2Pq_1 .$$

4. The plane which contains both  $n(q_1)$  and the line segment from  $q_0$  to  $q_1$  is given by [57],

$$[(q_1 - q_0) \times n(q_1)]^t \cdot (v - q_1) = q^t v = 0, \quad (11)$$

where "x" signifies the cross product. The simultaneous solution of (5) and (11) represents an arc connecting  $q_0$  to  $q_1$  along the surface. This arc can be transformed to two-dimensional coordinates to determine its direction vector, and its x- and y-interval bounds, by the following transformation.

$$z = \frac{q}{|q|} , \quad x = \frac{q_1 - q_0}{|q_1 - q_0|} ,$$

$$T = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot \\ x^t & (z \times x)^t & z^t & q_0 \\ \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Via this transformation, (5) becomes

$$v^t(T^t_{PT})v = v^t_Qv = 0, \quad (12)$$

and (11) becomes  $z=0$ . Thus, by plugging  $z=0$  into (12), a planar conic representation of the arc is obtained, giving the system depicted in Figure 25. Choose the smaller arc segment connecting  $p_0$  to  $p_1$  and call this  $arc_{01}$ . This arc, back in the piece coordinates, will contain the final arc which connects  $q_0$  to the boundary. In this two-dimensional system the x-interval and y-interval limits of the arc can be easily determined. Also, the direction vector  $r$  of  $arc_{01}$  at  $p_1$  can be obtained by choosing a vector along the tangent to  $arc_{01}$  at  $p_1$  which lies in the plane perpendicular to the surface normal.

5. Let  $i=1$ .
6. If  $arc_i$  of the boundary and  $arc_{01}$  intersect, excluding the intersection of  $arc_j$  and  $arc_{01}$  at the point  $q_1$ , then extract the subarc of  $arc_{01}$  from  $q_0$  to the point of intersection closet to  $q_0$  along  $arc_{01}$ . Let this subarc be the new  $arc_{01}$  connecting  $q_0$  to the boundary. Let the point of intersection of this new  $arc_{01}$  with the boundary be the new point  $q_1$ . Intersections between arcs can be

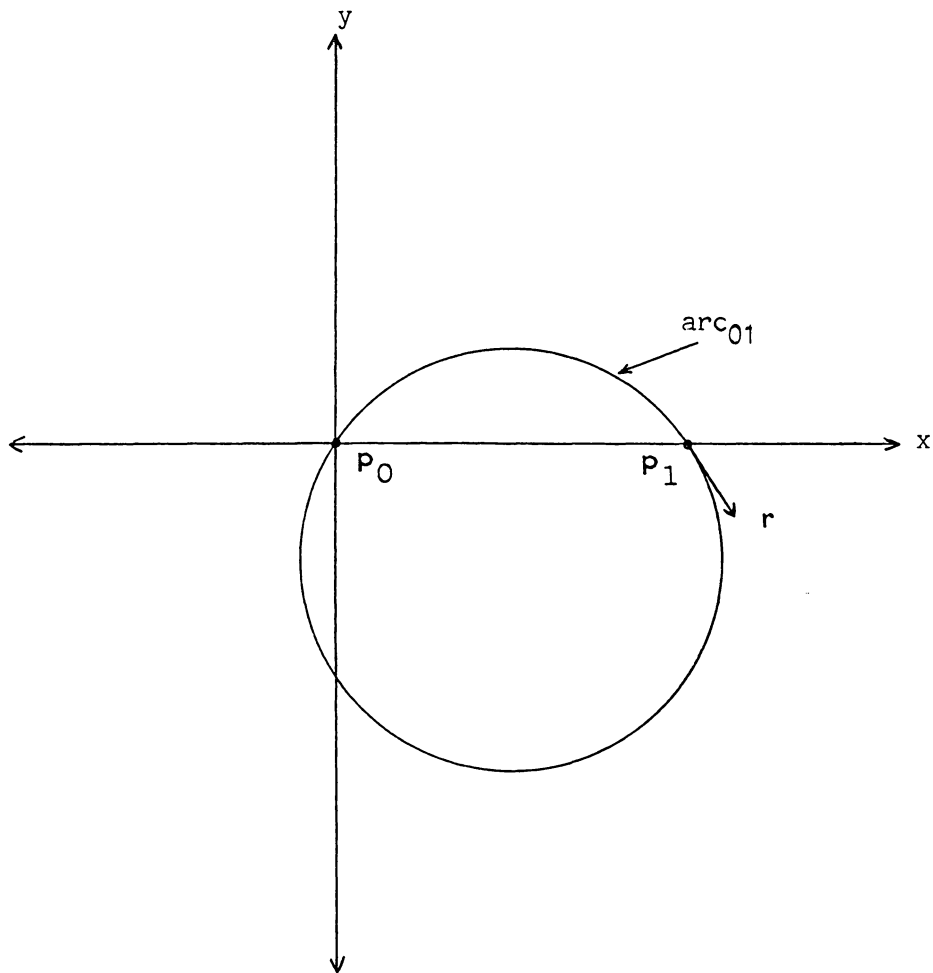


Figure 25. Planar Conic Arc

determined by transforming them into the same coordinate system and solving the resulting equations. To do this, a primary description of an arc may be thought of as the intersection of the  $z=0$  plane and a conic cylinder. This allows the techniques of pages 70-72 to be used to transform these two equations (and thus the arc) to any coordinate system. So the intersection problem becomes one of solving four simultaneous equations in three unknowns. This can be solved by standard algebraic methods.

7. Let  $i = i + 1$ ; if  $i \leq$  (number of arcs in the boundary) then go to step 6.
8. Arc<sub>01</sub> now connects the test point  $q_0$  to the boundary at point  $q_1$  on some arc, call it arc<sub>k</sub>. No part of the boundary intervenes between  $q_0$  and  $q_1$  along arc<sub>01</sub>. As in step 4, a direction vector  $r$  for arc<sub>01</sub> at point  $q_1$  can be determined. Similarly, a direction vector  $t$  for arc<sub>k</sub> at the point  $q_1$  can be obtained.
9. Let  $m = r \times t$ .
10. If  $m = 0$ , then the vectors  $r$  and  $t$  are collinear and the test for point-in-boundary can not be completed at this point. In this case, a new arc<sub>j</sub> may be chosen in step 1,

or a new point  $q_0$  chosen in step 2, and the subsequent steps repeated to generate new  $r$  and  $t$  vectors.

11. If  $m \cdot n(q_1) > 0$ , then the point  $q_0$  is inside the piece boundary, since the boundary is right-handed and the cross-product of  $r$  and  $t$  has given the same direction as the surface normal.
12. If  $m \cdot n(q_1) < 0$ , then the point  $q_0$  is outside the piece boundary.

Figure 26 illustrates the use of this algorithm for a point-in-polygon problem.

## A Discretized Representation

Once the outside RPP of a part has been determined, in order to obtain the tree decomposition, e.g. octree or R-tree, the part must be converted to a discretized representation. The part must be represented in a voxel array to facilitate the tree building algorithms.

This voxel array representation is obtained by dividing the bounding RPP of the part into a regular array of rectangular parallelepipeds (voxels) of the resolution size desired. For octrees, array elements will generally be cuboids. These cuboids must be of a certain size and aligned with the

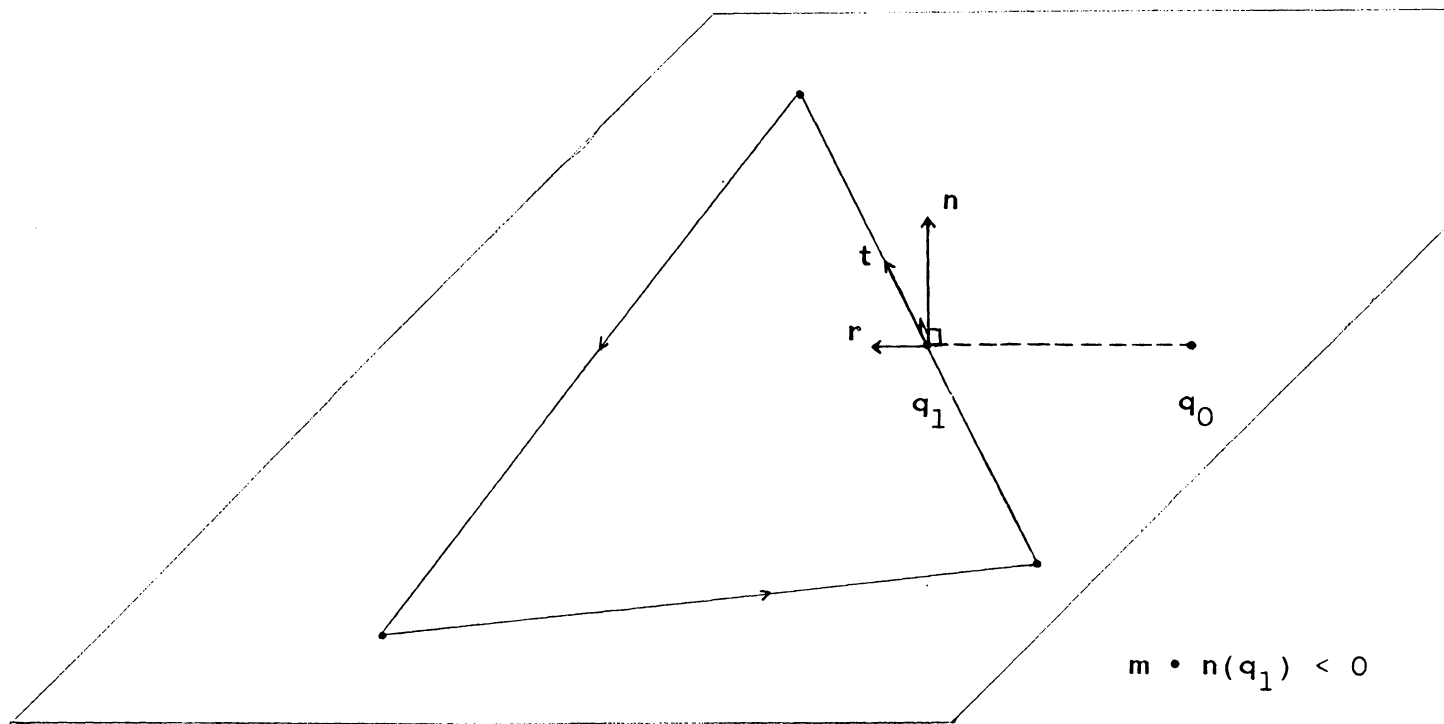


Figure 26. Point-in-Boundary Solution

voxel array of the universe cube. Thus the size and position of the bounding RPP is highly constrained for octrees, as is the resolution in each coordinate direction. The R-tree, however, is independent of the universe origin and can therefore take full advantage of the minimal bounding RPP found earlier. In addition, since the voxels for R-trees are not constrained to cuboids, different resolutions can be used for the three coordinate directions.

The number of divisions along each axis direction of the outside RPP determines the resolution of the voxel array. The part is represented in this array by voxels containing any of the interior of the part being labeled as interior voxels and all others being labeled exterior voxels. Once this has been achieved the algorithms of the remaining two sections of this chapter will be applicable.

The labeling of interior voxels first requires that the voxels that contain the part surfaces be labeled as boundary voxels. This is done by extending straight lines from the centers of the rectangular divisions of three faces of the outside RPP. Recall that the edges of the outside RPP are parallel to the world coordinate axes, so that the straight lines extended are parallel to the world coordinate axes also. The voxels in which these lines intersect the pieces of the surfaces of the part are labeled as boundary voxels. For example, consider Figure 27. This figure depicts the labeling of boundary voxels for one piece of a part surface.

Straight lines are extended from the centers of the rectangular divisions of one x-z face of the outside RPP. These lines, represented parametrically, can be transformed to Frame(piece) and there checked for intersection with the piece. The y-coordinate of the intersection in Frame(world) determines which voxel (x,y,z) is to be labeled as a boundary voxel. Since it is possible for an x-z line to miss the piece surface narrowly in some voxels, lines should also be extended from a x-y face and a y-z face, and tested in a similar manner. It will still be possible for the piece to intersect an unlabeled voxel, but since the volume of the intersection will now be much smaller than the resolution of the array, such intersections may be ignored.

During the labeling of the boundary voxels, the surfaces related to holes in the part may be included so that the voxel representation, and thus the trees, will contain this information. The hole information was not needed for the outside RPP calculations and is easily included here by adding a few surfaces to the total part description. This process of labeling boundary voxels is expensive time-wise, primarily because, when the primary description of a piece is encountered, the point-in-boundary algorithm must be invoked.

With the boundary voxels labeled, the interior and exterior voxels may be labeled by using a three-dimensional connected components algorithm [3] on the unlabeled voxels. The boundary voxels may then be relabeled to either interior or

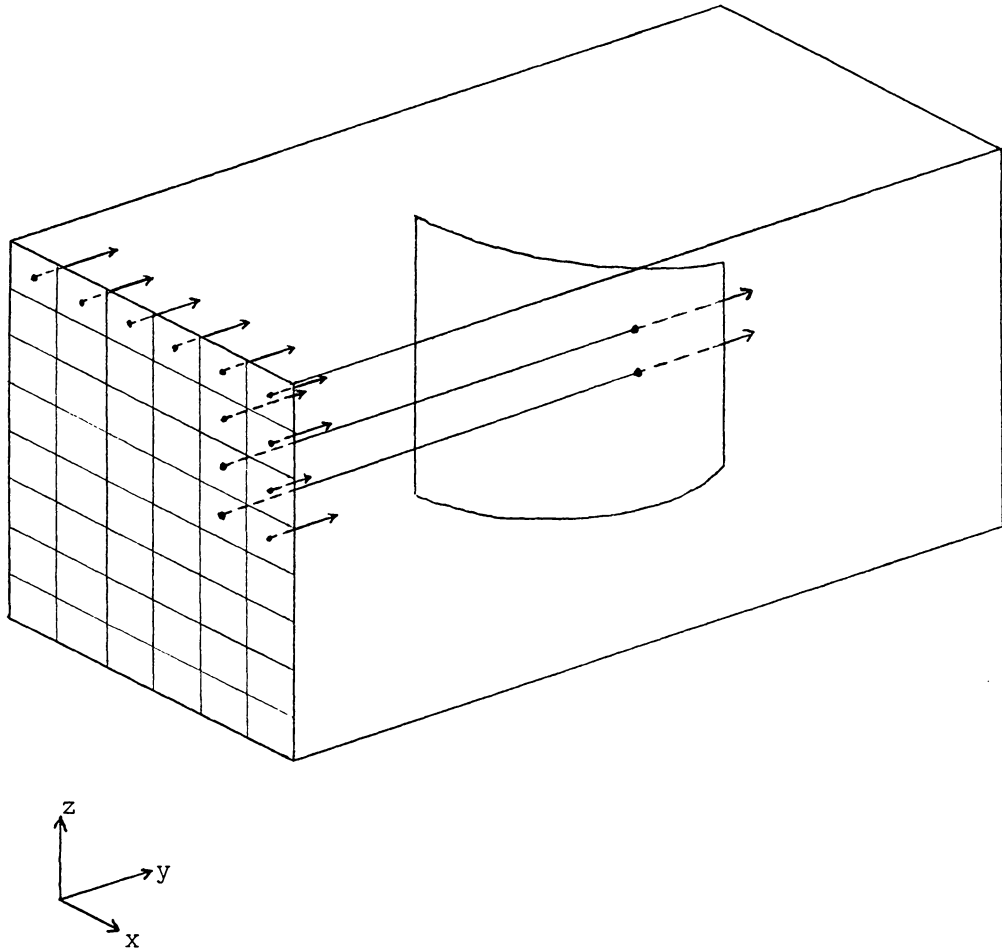


Figure 27. Single Piece Example of Discretization

exterior voxels, depending on which relabeling will produce the more desirable type of error, some of the part not being encoded or some free-space being encoded as a solid.

The process of discretizing the part is a very time-consuming process. It need be done only once for R-trees so long as the part is not rotated. It need be done only once for octrees also, so long as the part is not rotated, or translated a distance other than an integer multiple of the base cuboid size. If the part is translated an integer multiple of the base cuboid size, it is possible to rebuild the octree from the octree of the untranslated part. Thus, for many applications, the discretizing of the object can be considered a preprocessing task.

### The Inside RPP

The outside RPP has been determined and a discretized representation of the part obtained. It would seem that tree building algorithms are now in order. However, there is one major problem of the R-tree concept that has yet to be addressed. How is the inside RPP to be determined?

One technique for determining the inside RPP was presented by Lee, et. al. [31]. This algorithm used the three-dimensional distance algorithm, developed by Lee [30] from the two-dimensional case [47], in an extension of an earlier idea used in two-dimensional feasibility studies [21]. Un-

fortunately, this approach does not find the maximum volume inside RPP in all situations and therefore leads to less than optimal R-trees.

Another algorithm is presented here which always finds the maximum volume inside RPP for the given voxel array representation of a solid. This algorithm operates on the principle that the total number of possible inside RPP's is finite, and that the maximal RPP can be determined by a structured search over a relatively small number of these possible RPP's. In addition, this search can be conducted in a single pass of the array. This is an advantage when arrays are stored as images, i.e. row-by-row in auxiliary storage.

Consider the two-dimensional example in Figure 28. The reader should note that rectangles 1 and 2, although they are possible inside rectangles, should not be considered since there exists much larger rectangles which contain these smaller ones. In fact, it is possible to extend any inside rectangle which does not touch the boundary until it does touch the boundary, forming a larger inside rectangle. Therefore, the maximal inside rectangle must touch the boundary in at least two points.

Suppose, then, that the image is processed row-by-row. At each row, the boundary will delimit strings of continuous interior pixels. These strings of pixels give one maximum dimension of rectangles crossing the row. Substrings of these strings can be components of the maximal inside rectangle

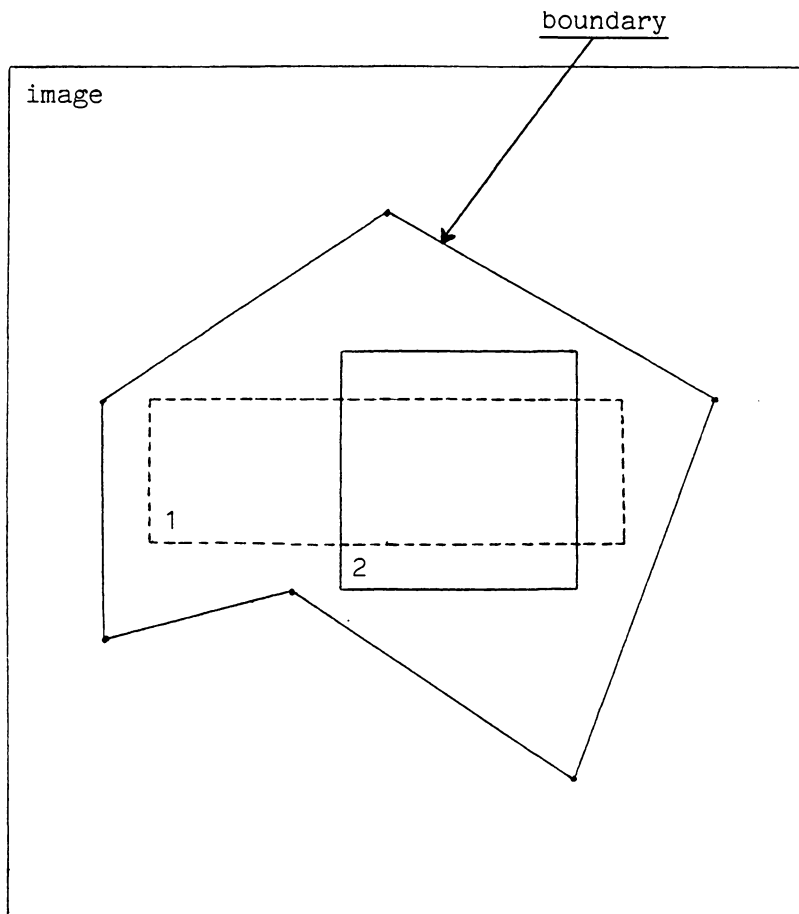


Figure 28. Possible Inside Rectangles.

only if they are delimited by the boundary on other rows. For example, Figure 29 depicts the row-by-row strings delimited by a two-dimensional boundary. The maximal inside rectangle is shown as a shaded region. Note that this rectangle is delimited by the boundary at row 17 and at row 75, and is elsewhere contained by the strings on each row.

This suggests the following algorithm for locating maximal inside rectangles in two-dimensional images. Maintain two lists, one for the rectangles being generated at the current row, and another for the rectangles of the previous rows. The rectangles of the previous rows list compose the set of all possible rectangles delimited by the boundary on previous rows, which can still be extended to the current row. The current row list, which is generated as the current row is processed, consists of rectangles created by the combination of the limits imposed by the continuous strings of interior pixels in the current row and the limits of the rectangles to be extended from the previous rows. Thus if the first and last column limits of a rectangle from the previous rows list is not contained by a string of interior pixels in the current row, the rectangle can not be extended further and has reached its maximal size. Any rectangle so completed is deleted from the lists and tested against the maximum area rectangle found so far.

After creating all the possible rectangles that can be extended into current row or that could begin at the current

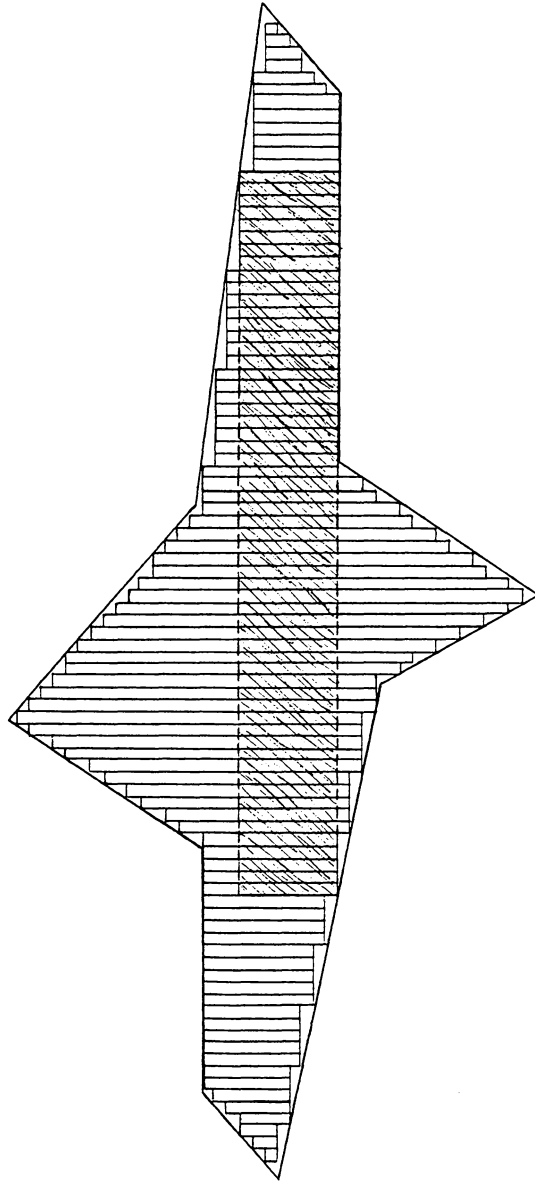


Figure 29. Row-by-Row Development of Inside Rectangle

row, and adding these to the current row list, the previous rows list is no longer needed and may be deleted. The current row list becomes the previous rows list upon moving to the next row in the image, and a new current row list will be generated. By maintaining this running list of possible maximal rectangles, the inside rectangle can be determined in one pass of the image. For the example of Figure 29, where the image size is  $100 \times 100$ , no more than 119 rectangles are examined by the algorithm.

For three-dimensional images, where the added dimension is created by having multiple-band images, the same ideas are applicable. The algorithm for the two-dimensional case can be used on the current band and the resulting rectangles compared with the RPP's to be extended from the previous bands. Those RPP's which can be extended, and those that are newly created by the limits imposed by the rectangles of the current band, are added to a current band list. Those which can not be extended are completed and compared with the maximal RPP encountered so far. As before, the current list becomes the new previous list and the image can be explored in one pass.

## Tree Building

Procedures for building octrees are well-known and are basically an extension of the quadtree building procedures

described by Samet [51,52]. The procedure for building an R-tree follows.

Given a solid represented by labeled voxels in a bounding RPP, such as that obtained as previously described, the inside RPP of the solid can be determined using the algorithm described in the preceding section. By extending the planes of the faces of the inside RPP to the outside RPP, the image will be segmented into 27 subregions, as shown in Figure 17. Those of these subregions which contain some interior voxels will constitute the children nodes of the outside RPP. The children nodes will be ordered in the final tree. The inside RPP, which is labeled "full", meaning it contains only interior voxels, is always the first child of the outside RPP.

Each child subregion will be examined according to some arbitrary order. If a subregion contains no part of the solid, i.e. no interior voxels, that subregion of the outside RPP need not be processed further. Otherwise, the subregion is treated as an outside RPP and the inside RPP for this child is determined by the same algorithm used before. This process is continued all the way down the tree until the desired resolution is reached.

Thus, the subtree for a child is completed before processing any of its siblings - the tree is built depth-first. The children at any node of the tree are ordered according to the size of their inside RPP's. The child with the largest

inside RPP will be the second child of the node, the inside RPP being the first.

Note that the R-tree must be built from the top-down, while the octree may be built from the bottom-up. This means that the R-tree building procedure could have a greater execution time. However, since the R-tree, for many types of solids, results in considerably fewer nodes, the difference in processing times may not be significant. Also, generating either type of tree is generally a time-consuming process so that this chore is usually delegated to a preprocessing phase where speed efficiency is less critical.

Now that the procedures for building R-trees have been established, some operations with these tree decompositions will be examined.

## CHAPTER 7 - OPERATIONS ON R-TREES AND OCTREES

For a robotic system, there are three operations on tree decompositions that are of particular importance. Since robots typically manipulate the objects in their environment, or objects are moved to facilitate inspection tasks, it is important to be able to translate and rotate the tree decomposition representations. Also, since collision checking is a prominent operation in planning robot movements, a system must be able to determine if two trees intersect.

This chapter will review the principles of these three operations with respect to R-trees and octrees.

### Translation

Several authors have presented algorithms for the translation of octrees [3,25,38]. These algorithms either require that translations occur only in integer multiples of the base voxel size [3], or that some error is incurred in the translated octree so that the original octree must always be stored for further translations [38]. In either case, translation of an octree requires rebuilding of the tree.

R-trees, however, are independent of the world coordinate system with respect to translation. The actual R-tree will be an integer decomposition of the discretized represen-

tation. That is, the RPP's that make-up the tree will be defined by the integers, first column, last column, first row, last row, first band, and last band, within the discretized outside RPP. This integer representation is less expensive space-wise than storing the actual real number positions of these RPP's. To locate the outside RPP a single real vector  $l$  is needed. If the R-tree is to be translated (by any amount,) the only modification is to  $l$ . Thus, translation of R-trees is trivial.

### Rotation

Arbitrary rotation of octrees is not possible, although some schemes have been presented for obtaining approximations to rotated trees [7,38]. Rotations by multiples of  $90^\circ$  are possible with octrees and several algorithms have been given [25,38]. These algorithms are based on the fact that a rotation by a multiple of  $90^\circ$  results in a reordering of the nodes of the tree. No new nodes are created and the entire operation can be carried out by a simple relabeling algorithm.

Since R-trees are composed of RPP's aligned with the world coordinate axes, as are octrees, they have the same properties with respect to rotations as do octrees. Arbitrary rotations of R-trees are not generally possible, though approximation schemes could be devised. Rotations by multiples of  $90^\circ$  are possible and require only that each node of

the tree be relabeled. For example, for a rotation of  $90^\circ$  about the x-axis, where columns of the outside RPP image segment the x-axis of the world coordinate system; the (first row, last row) pair of each node is swapped with the (first band, last band) pair of that node. If the outside RPP did not have an edge lying in the x-axis of the world coordinates, then the position vector  $l$  must also be updated to complete the operation. This is an advantage over the octree, which must be translated to the origin, rotated, and then translated back to complete the operation.

### Interference Detection

The algorithm for determining whether two objects represented by octrees have any voxels in common is well-known [7]. This algorithm is given in Figure 30, slightly modified to accommodate a generalized tree structure. The octrees tested by this algorithm are in the form described in Chapter 5. It is assumed that there is a routine READ which obtains a node of a tree from the data structure. As before, the number of accesses to the octree data structures is an indication of the amount of processing done by the algorithm.

Recall that the octrees of two objects share the same universe cube. Also, because the tree decomposition is a regular one, the positions of the eight children of the universe cube are the same for either object, as are the posi-

```

procedure OCTREES_INT( X, Y )

    // intersect the octree pointed to by X
    // with the octree pointed to by Y      //

    if ( X ≠ 0 ) and ( Y ≠ 0 ) then

        [ call READ( X, RPP_X, CHILD_X, SIBLING_X, STATUS_X )
          call READ( Y, RPP_Y, CHILD_Y, SIBLING_Y, STATUS_Y )
          if ( STATUS_X = void ) or ( STATUS_Y = void ) then
              OCTREES_INT = OCTREES_INT( SIBLING_X, SIBLING_Y )
          else
              [ if ( STATUS_X = full ) or ( STATUS_Y = full )
                then OCTREES_INT = true
                else
                    OCTREES_INT = OCTREES_INT( CHILD_X, CHILD_Y )

                if ( OCTREES_INT ≠ true ) then
                    OCTREES_INT = OCTREES_INT( SIBLING_X,
                                                SIBLING_Y ) ]
              ]
        ]

    else

        OCTREES_INT = false

    return

end OCTREES_INT

```

Figure 30. Octrees Intersection Algorithm

tions of the children of any octant in the tree. Thus, at all levels of the octrees of the two objects the node RPP's exactly correspond, that is, provided both octrees have nodes down to that level. This means that no routine is needed to determine if two RPP's intersect, and that the algorithm need only compare the TYPE fields of corresponding octants to determine if an intersection exists. However, this also means that the octree must retain both its white and black nodes, and therefore take-up more space.

Unfortunately, the R-trees of two objects do not share a universe RPP, nor are their nodes guaranteed to correspond in any uniform manner. This requires that a child of one R-tree, whose parent node intersects a node of another R-tree, may have to be compared with each child of the node of the other R-tree in order to determine an intersection. If no intersection is found for this child, then its sibling must be compared with each child of the intersecting R-tree node. This process will continue until each child of the first R-tree has been tested with every child of the second R-tree, (or until an intersection is found.) Thus, the R-tree algorithm has worse complexity due to the irregularity of the decomposition, but it should be remembered that the major factor determining the speed of processing is the nature of the two objects and their (non-)intersection.

The algorithm for detecting interference between two R-trees is given in Figure 31. As in earlier algorithms, the

subprograms READ and INTERSECTS are assumed. It is also assumed, for simplicity, that the array encodings of the R-trees have the same origin. If this was not the case, the RPP's of one tree would have to be translated before the use of the INTERSECTS function.

The reader should note that if the objects are far enough apart so that the bounding RPP's at the highest level of the two R-trees do not intersect, this will be detected immediately; whereas the octree intersection, depending on the size of the objects with respect to the size of the universe cube, may take a considerable amount of computation. The same is generally true if two objects overlap greatly. In this case, their inside RPP's will intersect, resulting again in early detection. For example, consider the exact alignment of the 64×64×64 images of Figures 18 and 19. The intersection of the R-trees of these objects, using the RTREES\_INT algorithm encoded in Rational FORTRAN, resulted in an intersection being detected after only four accesses to the data structures. The octree algorithm, however, required 40 accesses to the octree data structures.

For the other combinations of objects in Chapter 5 the following results were obtained. The intersection of the quadrilateral cylinder of Figure 19 and the rotated, cubic shell obtained from Figure 21 yielded an access count of four for the R-tree representations, and an access count of 44 for the octree representations. Again, the R-tree algorithm per-

```

procedure RTREES_INT( X, Y )
    // intersect two R-trees pointed to by X and Y //
    if ( X ≠ 0 ) and ( Y ≠ 0 ) then
        [ call READ( X, RPP_X, CHILD_X, SIBLING_X, FULL_X )
          call READ( Y, RPP_Y, CHILD_Y, SIBLING_X, FULL_Y )
          if INTERSECTS( RPP_X, RPP_Y ) then
              [ if ( FULL_X ) and ( FULL_Y ) then
                  RTREES_INT = true
                else if ( FULL_X ) then
                  RTREES_INT = RTREES_INT( X, CHILD_Y )
                else if ( FULL_Y ) then
                  RTREES_INT = RTREES_INT( Y, CHILD_X )
                else
                  RTREES_INT = RTREES_INT( CHILD_X, CHILD_Y )
              ]
          if ( RTREES_INT ≠ true ) then
              RTREES_INT = RTREES_INT( X, SIBLING_Y )
          if ( RTREES_INT ≠ true ) then
              RTREES_INT = RTREES_INT( SIBLING_X, Y )
          ]
        else
            RTREES_INT = false
        return
    end RTREES_INT

```

Figure 31. R-trees Intersection Algorithm

forms well in this case, due to the compactness of the R-tree representation for the quadrilateral cylinder. A less compact R-tree representation, such as that of the four-sided polyhedron of Figure 18, intersected with the cubic shell results in a poorer performance by the R-tree. Detection of interference between the polyhedron and cubic shell required 26 accesses for the R-trees and 82 accesses for the octrees.

From these results it would appear that R-tree interference detection is faster than octree interference detection. This is generally true when there is a fair amount of overlap between the objects, or when the objects have fairly space-efficient R-tree representations. However, consider the intersection between the polyhedron of Figure 18 and the object of Figure 32. Here, there is not much overlap between two objects with rather inefficient R-tree representations. Interference between these objects was detected by the R-tree algorithm after 124 accesses to the R-trees, and by the octree algorithm after 40 accesses to the octrees.

The real deficiency of the R-tree algorithm is the worst case performance. The octree algorithm will never require more accesses than twice the number of nodes in the smaller octree. The R-tree algorithm, on the other hand, can make several times as many accesses to the data structures as there are nodes in the largest R-Tree. As an example, consider the object of Figure 32 shifted slightly to the position shown in Figure 33. This shifted object now narrowly

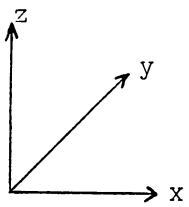
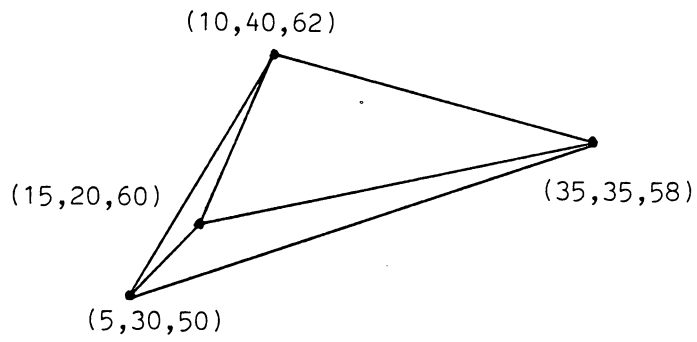


Figure 32. Small Polyhedron

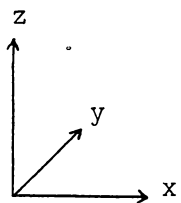
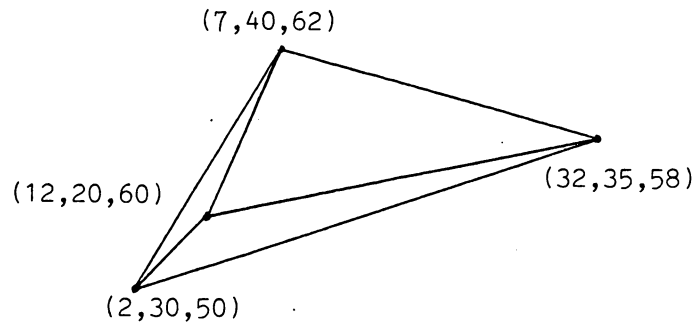


Figure 33. Shifted Small Polyhedron

misses intersecting the polyhedron of Figure 18. The R-tree algorithm determines that there is no intersection only after 29,630 accesses to the R-trees.

## CHAPTER 8 - CONCLUSIONS

This thesis has shown that an irregular decomposition can have some advantages over a regular decomposition of space. These include better space efficiency and less representation variation under rigid body transformations. The principle disadvantage of an irregular decomposition is that interference detection between two irregularly decomposed objects can require much greater time-complexity. However, the performance of both types of decompositions is highly data dependent.

In addition, this thesis has shown how to obtain tree decompositions from certain types of boundary and CSG representations. In particular, the sections of Chapter 6 involved with finding the bounding RPP of a part and obtaining discretized representations, are the central steps needed for obtaining both octrees and R-trees from these other representations.

For the inspection scenario, it is the author's opinion that R-trees would prove more useful than a straight octree encoding. The octree for a machined object the size of an aircraft bulkhead would be prohibitively large if built to accommodate a resolution sufficient for the inspection task. In addition, the octree would not be readily translatable, as mentioned earlier in this thesis. These two factors over-

shadow the poorer performance of R-trees in interference detection. Also, remember that the bulkhead is divided into parts which are the rather blockish structures of ribs, webs, and flanges. The regularity of many of these shapes with respect to the world coordinate axes would lead to relatively simple R-trees, such as those obtained for the quadrilateral cylinders of Chapter 5. Interference detection between the robot arms and a set of small R-trees could be done quickly, and the exact representation of the relational model could be indexed by the trees for fine inspection tasks.

Hopefully, this thesis has demonstrated that the current status of object representation is not the final word on the subject, as some seem to believe. There are still many avenues to explore. Some areas in which additional research seems warranted include a more general comparison of regular and irregular decompositions, the wide-spread introduction of mathematical topology into the problems of object representation, and an exploration of other combinations of regular and irregular decompositions for use in robotic systems.

## BIBLIOGRAPHY

1. Aggarwal, J. K., Davis, L. S., Martin, W. N., and Roach, J. W., "Survey: Representation Methods for Three-dimensional Objects," in Progress in Pattern Recognition, Kanal, L. N., and Rosenfeld, A., eds., North-Holland, New York, 1981.
2. Agoston, M. K., Algebraic Topology: A First Course, Marcel Dekker, Inc., New York, 1976, p 286.
3. Ahuja, N., and Nash, C., "Octree Representations of Moving Objects," Computer Vision, Graphics, and Image Processing 26, Number 2, May 1984.
4. Ballard, D. H., "Strip Trees: A Hierarchical Representation for Curves," Communications of the ACM, Volume 24, Number 5, May 1981.
5. Ballard, D. H., and Brown, C. M., Computer Vision, Prentice-Hall, Englewood Cliffs, New Jersey, 1982, pp 264-283
6. Blum, H., "A Transformation for Extracting New Descriptors of Shape," in Models for the Perception of Speech and Visual Form, Dunn, W., ed., Cambridge, Mass., MIT Press, 1964.
7. Boaz, M. N., Spatial Coordination of Transfer Movements in a Dual Robot Environment, Master's Thesis, Virginia Tech, 1983.
8. Boyse, J. W., "Interference Detection Among Solids and Surfaces," Communications of the ACM, Volume 22, Number 1, January 1979.
9. Burton, W., "Representations of Many-Sided Polygons and Polygonal Lines for Rapid Processing," Communications of the ACM, Volume 20, Number 3, March 1977.
10. Engineer, S. N., An Experimental Spatial Information System, Master's Thesis, Virginia Tech, 1983.
11. Faugeras, O. D., and Ponce, J., "Prism Trees: A Hierarchical Representation for 3-D Objects," Proceedings of International Joint Conference on

Artificial Intelligence, Karlsruhe, West Germany, August 1983, pp. 982-988.

12. Ferrari, L., Sankar, P. V., and Sklansky, J., "Minimal Rectangular Partitions of Digitized Blobs," Computer Vision, Graphics, and Image Processing 28, Number 1, October 1984.
13. Finkbeiner, II, D. T., Introduction to Matrices and Linear Transformations, W. H. Freeman and Company, San Francisco, 1966, p 207.
14. Fitzgerald, W., Gracer, F., and Wolfe, R., "GRIN: Interactive Graphics for Modeling Solids," IBM Journal of Research and Development, Volume 25, Number 4, July 1981.
15. Forrest, A. R., "On Coons and Other Methods for the Representation of Curved Surfaces," Computer Graphics and Image Processing 1, Number 4, December 1972.
16. Gargantini, I., "An Effective Way to Represent Quadtrees," Communications of the ACM, Volume 25, Number 12, December 1982.
17. Gargantini, I., "Linear Octrees for Fast Processing of Three-Dimensional Objects," Computer Graphics and Image Processing 20, Number 4, December 1982.
18. Goldman, R. N., "An Urnful of Blending Functions," IEEE Computer Graphics and Applications, Volume 3, Number 7, October 1983.
19. Greville, T. N. E., ed., Theory and Applications of Spline Functions, Academic Press, New York, 1969, p 22.
20. Hacker, R., "Certification of Algorithm 112: Position of a Point Relative to a Polygon," Communications of the ACM, Number 12, December 1962.
21. Haralick, R. M., Fiala, J. C., and Shapiro, L. G., "Volume Intersection Problems," Technical Report, Spatial Data Analysis Laboratory, Virginia Tech, December 1983.
22. Haralick, R. M., Personal Communication, March 1984.
23. Hopcraft, J. E., Schwartz, J. T., and Sharir, M., "Efficient Detection of Intersections among Spheres," International Journal of Robotics Research, Volume 2, Number 4, Winter 1983.

24. Hunter, G. M., and Steiglitz, K., "Operations on Images Using Quad Trees," IEEE Transactions on Pattern Analysis and Machine Intelligence 1, Number 2, April 1979.
25. Jackins, C. L., and Tanimoto, S. L., "Octrees and Their Use in Representing Three-Dimensional Objects," Computer Graphics and Image Processing 14, Number 3, November 1980.
26. Jackins, C. L., and Tanimoto, S. L., "Quad-trees, Oct-trees, and K-trees: A Generalized Approach to Recursive Decomposition of Euclidean Space," IEEE Transactions on Pattern Analysis and Machine Intelligence 5, Number 5, September 1983.
27. Klinger, A., and Dyer, C. R., "Experiments on Picture Representation Using Regular Decomposition," Computer Graphics and Image Processing 5, Number 1, March 1976.
28. Krusemark, S. W., A Transportable Image Processing System Called GIPSY, Master's Thesis, Virginia Tech, 1983.
29. Krusemark, S., and Haralick, R. M., "Achieving Portability in Image Processing Software Packages," Proceedings IEEE Conference on Pattern Recognition and Image Processing, Las Vegas, Nevada, June 1982, pp. 451-467.
30. Lee, S. J., "Two-Pass Three-Dimensional Distance Determination," Technical Report, Spatial Data Analysis Laboratory, Virginia Tech, July 1984.
31. Lee, S. J., Zhuang, X., and Fiala, J. C., "The Three-Dimensional R-tree," Technical Report, Spatial Data Analysis Laboratory, Virginia Tech, August 1984.
32. Levin, J. Z., "Mathematical Models for Determining the Intersections of Quadric Surfaces," Computer Graphics and Image Processing 11, Number 1, September 1979.
33. Li, M., Grosky, W. I., and Jain, R., "Normalized Quadtrees with Respect to Translations," Computer Graphics and Image Processing 20, Number 1, September 1982.
34. Lin, W. C., and Fu, K. S., "A Syntactic Approach to 3-D Object Representation," IEEE Transactions on Pattern Analysis and Machine Intelligence 6, Number 3, May 1984.

35. Lumia, R., "A New Three-Dimensional Connected Components Algorithm," Computer Vision, Graphics, and Image Processing, Number 2, August 1983.
36. Mahl, R., "Visible Surface Algorithm for Quadric Patches," IEEE Transactions on Computers 21, Number 1, January 1972.
37. Marsden, J. E., and Tromba, A. J., Vector Calculus, W. H. Freeman and Company, New York, 1981, p 232.
38. Meagher, D., "Geometric Modeling Using Octree Encoding," Computer Graphics and Image Processing 19, Number 2, June 1982.
39. Nackman, L. R., and Pizer, S. M., "Three Dimensional Shape Description Using the Symmetric Axis Transform I: Theory," IEEE Transactions on Pattern Analysis and Machine Intelligence 7, Number 2, March 1985.
40. Newman, W. M., and Sproull, R. F., Principles of Interactive Computer Graphics, Second Edition, McGraw-Hill Book Co., New York, 1979, pp 309-330.
41. O'Rourke, J., and Badler, N., "Decomposition of Three-Dimensional Objects into Spheres," IEEE Transactions on Pattern Analysis and Machine Intelligence 1, Number 3, July 1979.
42. Paul, R. P., Robot Manipulators: Mathematics, Programming, and Control, MIT Press, Cambridge, Mass., 1981.
43. Pfaltz, J. L., and Rosenfeld, A., "Computer Representation of Planar Regions by Their Skeletons," Communications of the ACM, Volume 10, Number 2, February 1967.
44. Requicha, A., "Representations for Rigid Solids: Theory, Methods, and Systems," Computing Surveys 12, Number 4, December 1980.
45. Requicha, A. A. G., and Voelcker, H. B., "Solid Modeling: A Historical Summary and Contemporary Assessment," IEEE Computer Graphics and Applications, Volume 2, Number 2, March 1982.
46. Requicha, A. A. G., and Voelcker, H. B., "Solid Modeling: Current Status and Research Directions," IEEE Computer Graphics and Applications, Volume 3, Number 7, October 1983.

47. Rosenfeld, A., and Pflatz, J. L., "Sequential Operations in Digital Picture Processing," Journal of the ACM, Volume 13, Number 4, 1966.
48. Roth, S. D., "Ray Casting for Modeling Solids," Computer Graphics and Image Processing 18, Number 2, February 1982.
49. Rubin, S. M., and Whitted, T., "A 3-Dimensional Representation for Fast Rendering of Complex Scenes," Computer Graphics, Volume 14, Number 3, July 1980.
50. Saloman, K. B., "An Efficient Point-in-Polygon Algorithm," Computers and Geosciences, Volume 4, Number 2, 1978.
51. Samet, H., "Region Representation: Quadtrees from Binary Arrays," Computer Graphics and Image Processing 13, Number 1, May 1980.
52. Samet, H., "An Algorithm for Converting Rasters to Quadtrees," IEEE Transactions on Pattern Analysis and Machine Intelligence 3, Number 1, January 1981.
53. Samet, H., "A Quadtree Medial Axis Transform," Communications of the ACM, Volume 26, Number 9, September 1983.
54. Samet, H., "The Quadtree and Related Hierarchical Data Structures," CS-TR-1329, Computer Science Dept. and Center for Automation Research, University of Maryland, November 1983.
55. Sarraga, R. F., "Algebraic Methods for Intersections of Quadric Surfaces in GMSOLID," Computer Vision, Graphics, and Image Processing 22, Number 2, May 1983.
56. Sederberg, T. W., Anderson, D. C., and Goldman, R. N., "Implicit Representation of Parametric Curves and Surfaces," Computer Vision, Graphics, and Image Processing 28, Number 1, October 1984.
57. Selby, S. M., ed., CRC Standard Mathematical Tables, The Chemical Rubber Co., Cleveland, Ohio, 1972, p 559.
58. Shapiro, L. G., and Haralick, R. M., "A General Spatial Data Structure," Proceedings IEEE Conference on Pattern Recognition and Image Processing, Chicago, Illinois, May 1978, pp. 238-249.

59. Shapiro, L. G., Moriarty, J. D., Haralick, R. M., and Mulgaonkar, P. G., "Matching Three-Dimensional Models," Proceedings IEEE Pattern Recognition and Image Processing Conference, Dallas, Texas, August 1981, pp 534-541.
60. Shapiro, L. G., and Haralick, R. M., "A Hierarchical Relational Model of the Bulkhead," Preliminary Report to Martin Marietta Corporation, Dept. of Computer Science, Virginia Tech, December 1983.
61. Shimrat, M., "Algorithm 112: Position of Point Relative to Polygon," Communications of the ACM, Volume 5, Number 8, August 1962.
62. Stoll, R. R., Linear Algebra and Matrix Theory, McGraw-Hill, New York, 1952, p 111.
63. Tamminen, M., "Comment on Quad- and Octtrees," Communications of the ACM, Volume 27, Number 3, March 1984.
64. Tilove, R. B., "Set Membership Classification: A Unified Approach to Geometric Intersection Problems," IEEE Transactions on Computers 29, Number 10, October 1980.
65. Vaidya, P. D., An Experimental Spatial Information System, Master's Thesis, Virginia Tech, 1981.
66. Warnock, J. E., "A Hidden-Surface Algorithm for Computer Generated Half-Tone Pictures," TR 4-15, Computer Science Dept., University of Utah, June 1969.
67. Watson, L., Technical Report on B-splines, Research Meeting, Spatial Data Analysis Laboratory, Virginia Tech, October 1983.
68. Weiss, R. A., "BEVISION, A Package of IBM 7090 FORTRAN Programs to Draw Orthographic Views of Combinations of Plane and Quadric Surfaces," Journal of the ACM, Volume 13, Number 2, April 1966.
69. Wesley, M. A., Lozano-Perez, T., Lieberman, L. I., Lavin, M. A., and Grossman, D. D., "A Geometric Modeling System for Automated Mechanical Assembly," IBM Journal of Research and Development, Volume 24, Number 1, January 1980.
70. Yau, M., and Srihari, S. N., "Recursive Generation of Hierarchical Data Structures for Multidimensional Digital Images," Proceedings IEEE Conference on Pattern

Recognition and Image Processing, Dallas, Texas, August 1981, pp. 42-44.

71. Yau, M., and Srihari, S. N., "A Hierarchical Data Structure for Multidimensional Digital Images," Communications of the ACM, Volume 26, Number 7, July 1983.
72. Zhuang, X., "Parallelepiped Containing a Part," Technical Report, Spatial Data Analysis Laboratory, Virginia Tech, January 1984.

**The vita has been removed from  
the scanned document**