AN ALGORITHM FOR POLYHEDRON MODELLING AND ITS IMPLEMENTATION

by

KWAI-BIU RICKY YEUNG

B.Sc., The University of Texas at Arlington, 1979

M.Sc., Stanford University, 1980

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF

THE REQUIRMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

(Department of Computer Science)

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

April, 1984

© Kwai-biu Ricky Yeung, 1984

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

Department of Computer Science

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

Date April 25, 1984

ABSTRACT

This thesis describes an algorithm for calculating the theoretic set operations union, intersection, and difference of two polyhedra. The polyhedra are in the Eulerian surface description format, specified by their vertices, edges, and faces. The representation of polyhedra is based on the unambiguous Boundary Representation scheme. The domain of this algorithm includes all flat-surfaced polyhedra; however, nested holes and curved surfaces are not permitted.

The thesis is divided into three parts. The first part presents some mathematical definitions and the representation scheme for polygons and polyhedra. Since there are many similarities between 2-D polygon modelling and 3-D polyhedron modelling, the 2-D polygon modelling algorithm is presented next, followed by the 3-D polyhedron modelling algorithm.

TABLE OF CONTENTS

LIST OF FIGURES AND TABLE

ACKNOWLEDGEMENTS

I would like to thank my wife for her continued spiritual support and affectionate care while I studied at UBC. I am also grateful to my supervisor, Dr. G. F. Schrack, for his advice and consistent guidance.

CHAPTER 1

INTRODUCTION

Three-dimensional solid geometry plays an important role in the design and production of mechanical parts and other discrete objects. The application of computers to solve computational geometry problems is widely recognized.

Many of the current Computer Aided Design (CAD) systems and graphic programs do not have a good representation scheme for solids. Most current CAD systems generate solids by using the wireframe method. Although this approach is quite useful in representing objects, it has several deficiencies. The most obvious one is that wireframes are ambiguous. Figure 1 illustrates a frame that does not represent a unique polyhedron. Other ambiguous frame examples are provided by Markowsky and Wesley [8]. Furthermore, Requicha outlined several other deficiencies [12].

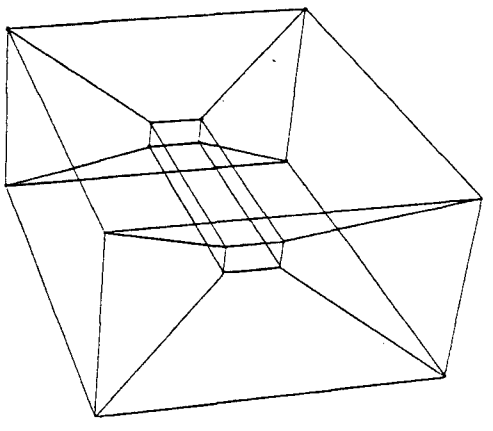

Figure 1. Wireframe ambiguity.

The LIG6 language and programming system [13] developed at The University of British Columbia uses the primitive "face" to construct solids. This program, however, can not detect the situation of two solids colliding with each other. Hence, a better representation scheme is desired to solve such ambiguities.

Solid modelling is a theory for the representation of solids preferable to wireframes because it is unambiguous and complete. There are six different schemes as categorized by Requicha [11]: (1) primitive instancing; (2) spatial enumeration; (3) cell decomposition; (4) constructive solid geometry; (5) boundary representation; and (6) sweeping.

Over the past ten years, several geometric modelling systems (GMS) have been developed which were based on the solid modelling theories above, using the technology of the 1970's. Constructive Solid Geometry (CSG) and Boundary Representation (B-Rep) have been the most popular schemes among these systems. CSG represents a solid by a tree of set operations and rigid motions. On the other hand, B-Rep defines a solid by a collection of vertices, edges, and faces and their neighourhood relationships. However, more recently, there is a new scheme using a different approach. It is the Octree scheme [9] which records the regions of space occupied by a solid at a fixed maximal resolution [5].

Most systems use CSG, B-Rep or a combinations of schemes as their primary representation scheme for input and output. For example, the system PADL-2 [2], developed at the University

of Rochester, uses CSG as its primary representation scheme while GMSolid [1], developed at General Motors, uses a hybrid CSG/is B-Rep scheme. Another system is BUILD-2 [3], developed by a group from Cambridge University, it uses B-Rep as its primary representation scheme. The domain of this thesis will be using the B-Rep as the representation scheme as well.

B-Rep was chosen as the primary representation scheme because all rectilinear polyhedra can be represented by this scheme. Besides, it is one of the unambiguous schemes for representing 3-dimensional solid objects. Furthermore, it covers a rich set of domains as in the schemes of cell decomposition or CSG. Finally, the available representations of vertices, edges, and faces ease the generation of line drawings and graphic objects, to support graphic interaction and other purposes as well [11].

GMSs using the solid modelling technique have many applications. In graphics, it can generate wireframe drawings, drawings with removed hidden lines and it can provide shaded renderings as well. In design analysis, mass properties can be calculated easily; interference detection, such as whether two parts are occupying the same space when fitting them together, is possible. In manufacturing, it can be used for NC verification, for instance, finding out the volume of a part that is cut off from a solid. Other applications such as the automatic generation of meshes for finite-element analysis, and the automatic generation of NC programs are in the research stage [12].

The objective of this thesis is to implement an algorithm for solid modelling where the domain covers all polyhedra whose faces are convex and/or concave polygons as defined in Chapter Two.

CHAPTER 2

DEFINITIONS AND REPRESENTATIONS

2.1 Polygon

2.1.1 Definition

A polygon (Figure 2) is the figure formed by choosing a sequence of n distinct points A(1),...,A(n), joining adjacent points with a line segment, and joining the last point with the first one. The points A(1),...,A(n) are the vertices of the polygon, and the segments A(1)A(2),...,A(n)A(1) are its sides (a vertex bounds exactly two sides) [7].

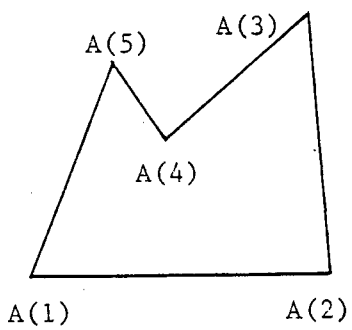

Figure 2. A planar, simple and straight polygon.

In this thesis, it is assumed that a polygon is planar (that is, all its sides lie in one plane), simple (that is, its sides intersect only at common bounding vertices), and straight (that is, all its sides are segments of straight lines.)

2.1.2 Representation

A planar, simple, and straight polygon P can be represented by a set of distinct points, (A(1),...,A(n)) where the points are ordered in a cyclic manner and the

vertices are given by its x, y, and z coordinates. The set of the bounding edges of the polygon is (A(1)A(2),...,A(n)A(1)).

2.2 Polyhedron

2.2.1 Definition

A polyhedron is a set of simple polygons (as defined above) with the property that every side of a polygon is also a side of exactly one other polygon in the system. The polygons are called the faces of the polyhedron, their sides are its edges, and their vertices are its vertices. The system as a whole is called the surface of the polyhedron [7].

In addition, in this thesis, well-formedness is assumed to be a global characteristic of the polyhedron; that is, polyhedra surfaces are made of connected and orientable faces, and the points shared by any pair of their faces are only those common edges or vertices; they do not self-intersect.

2.2.2 Representation

A well-formed polyhedron S is represented by a set of faces (F(1),...,F(n)). Each face F(i), i=1,...,n, is represented by an ordered set of edges E, each in turn constituting a cycle on F(i). Each edge E(j) of S, j=1,...,r, is defined by two endpoints P(k1) and P(k2) where P(k) = (X(k), Y(k), Z(k)), k=1,...,m, is a vertex of S.

2.3. Regularity

This thesis primarily deals with objects that are subsets of Euclidean spaces of one, two, and three dimensions. Tilove

[15] gave the following topological definitions of interior, closure, boundary, and regularity:

A point p is an element of the <u>interior</u> of a set X, iX, if there exists a neighourhood of p (e.g., an open ball about p) that is contained in X;

p is an element of the <u>closure</u> of X, kX, if every neighourhood of p contains a point of X;

p is an element of the <u>boundary</u> of X, bX, if p is an element of both kX and k(cX), where cX denotes the usual complement of X; and

a set X is said to be <u>regular</u> iff X = kiX.

An object from a regular set must not have dangling faces or dangling edges. Figure 3 shows an object which is not in the regular set. Furthermore, the regularization of a set is defined as rX = kiX.

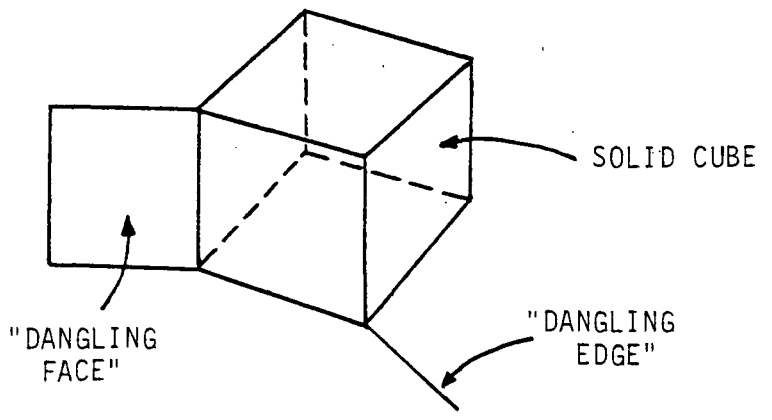

Figure 3. Non-regular object.

Under conventional set operators intersection (I), union (U), and difference (-), regular sets are not closed. However, the regularized set operators, I*, U*, and -*, do preserve regularity. Using the above topological notions, if X and Y are

two regular sets, the regularized set operations of X and Y can be defined as follows:

$$X \; I^* \; Y = ki(X \; I \; Y),$$

$$X \; U^* \; Y = ki(X \; U \; Y),$$

$$X \; -^* \; Y = ki(X - Y),$$

$$c^*X = kicX.$$

It can be shown that the regular sets and the regularized set operators form a Boolean algebra. Figure 4 illustrates an example that shows the difference between conventional and regularized intersection [11], [15].

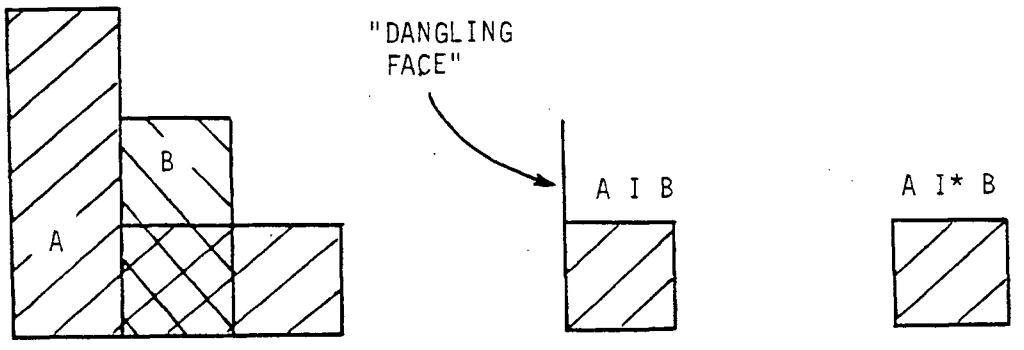

Figure 4. Conventional versus regularized intersection of two regular sets (sets shown in orthographic projection).

2.4. Set Membership Classification

Tilove [15] defines a <u>membership classification function</u> (Figure 5) as follows:

$$M[X,S] = (XinS, \; XonS, \; XoutS)$$

where

$$XinS = X \; I^{*\prime} \; iS = k'i'(X \; I \; iS)$$

$$XonS = X \; I^{*\prime} \; bS$$

$$XoutS = X \; I^{*\prime} \; cS$$

and S is a reference regular set in a subspace W of E^3 (W may be E^3 or a subset of E^3), and X is a candidate regular set in a subspace W' of W to be classified with respect to S. Note: primed symbols (e.g., k', and I*') denote operations in W', and unprimed symbols denote operations in W.

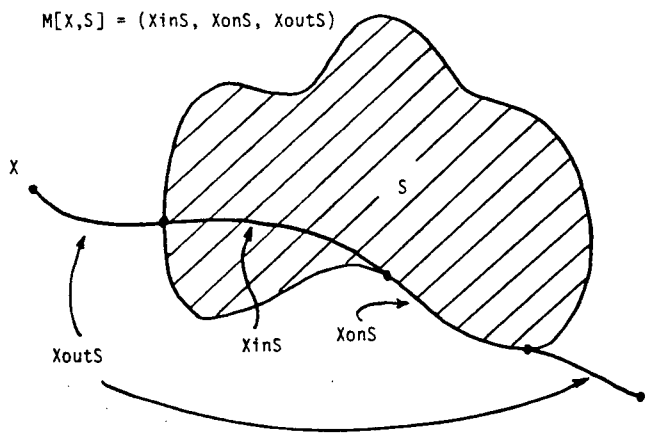

Figure 5. Membership classification function.

He also provided examples for different cases of this function as shown in Figure 6 and Figure 7 and claimed that the theory is valid for any choice of W and W'.

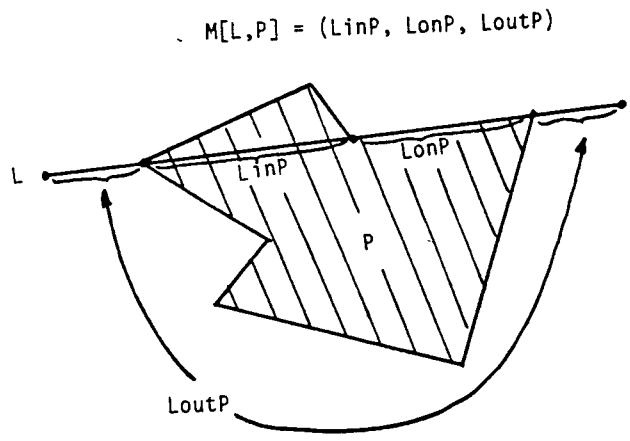

Figure 6. Membership classification example:
Line/polygon clipping.

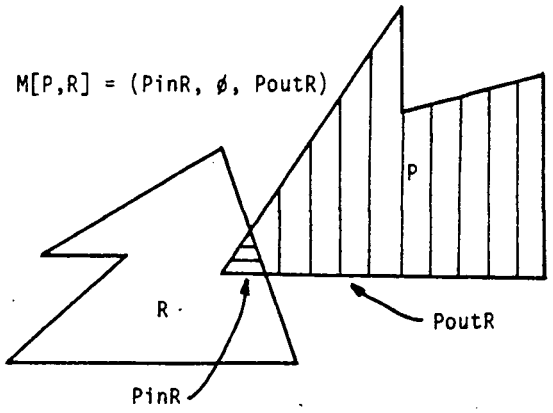

Figure 7. Membership classification example: Polygon intersection.

Results from the classification function will separate the candidate set into three quasi-disjoint decomposition parts, XinS, XonS, and XoutS. That means, the union of the three produces the candidate set, while the regularized intersection of the three is null [15].

S = A U* B

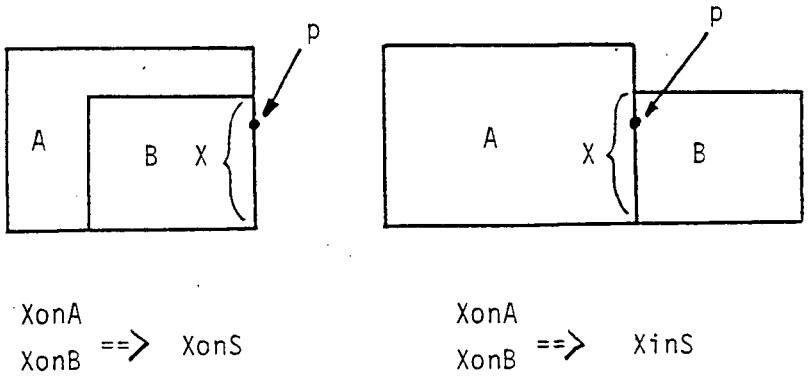

Figure 8. On/on ambiguity.

However, there exists an "On/on" ambiguity in classification as illustrated in Figure 8. The On/on ambiguity can be resolved into two different "On" types by testing the

neighbourhoods of a point p on X (see Appendix B for an algorithm). It is necessary to distinguish the two different types as follows:

<u>Xtype1onS</u> is that portion of space on X with the regular sets containing X on the same side of the space.

<u>Xtype2onS</u> is that portion of space on X with the regular sets containing X on opposite sides of the space.

Hence, the "on" edge of the object to the left of Figure 8 is a "Xtype1onS" while the right hand object is of type "Xtype2onS".

	Regularized set operations desired			
Segments	A I* B	A U* B	A -* B	B -* A
XinB	+			+
XinA	+		+	
Xtype1onB	+	+		
Xtype2onB			+	+
XoutB		+	+	
XoutA		+		+

+ : denotes included

Table I. Edge/Face segments to be included in the desired regularized set operations.

Thus, the classification function M[X,S] can be augmented as M'[X,S] = (XinS, Xtype1onS, Xtype2onS, XoutS) for sets for which "On/on" ambiguities can occur. Table I shows the relationship between the membership classification function and the regularized set operations of two polygons/polyhedra A and

B [5]. Depending on the desired operation, appropriate subsets of edge/face segments are chosen. Note that Xtype1onA is equal to Xtype1onB, as is Xtype2onA equal to Xtype2onB. It follows that there are six sets of segments, namely, XinA, XinB, Xtype1onA, Xtype2onB, XoutA, and XoutB.

CHAPTER 3

2-D POLYGON MODELLING ALGORITHM

3.1 Domain of Algorithm

The domain of this Polygon Modelling Algorithm is the set of all 2-dimensional planar, simple, and straight polygons. Moreover, two polygons which touch at a point are considered to be two distinct polygons (Figure 9).

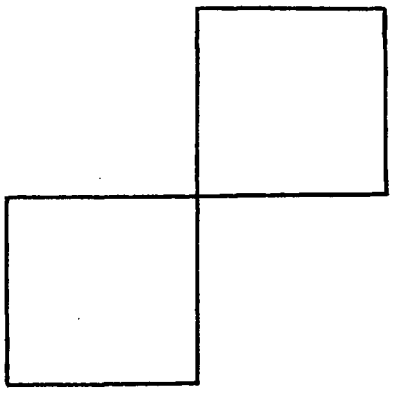

Figure 9. Polygons which touch at one point.

3.2 Data Structure

The following Pascal TYPEs illustrate the data structure of a polygon and a list of polygons:

```
XYpoint = RECORD x, y : Real END;  { (x,y) point }
PointList = @Vertex;   { order list of points }
Vertex = RECORD
            p : XYpoint;
            next : PointList
         END;
```

```
aPolygon = @polygon;          { a polygon }
polygon = RECORD
            num : integer;     { number of vertices }
            point : PointList { points of polygon  }
          END;
Polygons = @Lines;          { a list of polygons }
Lines = RECORD
          num : integer;     { number of vertices }
          next : Polygons;  { link to next one   }
          pt : PointList    { points of polygon  }
        END;
```

Polygon is a record of two components. The first component of the record, num, records the number of vertices in the polygon; the second component, point, is a linked list of (x,y) vertices of the polygon. Thus, a set of polygons is a linked list of polygons.

A line is represented by its endpoints. A list of line segments for the classifications XinB, Xtype1onB, Xtype2onB, and XoutB is represented as follows:

```
LineList = @Vertices;       { a list of lines }
Vertices = RECORD
             p, q : XYpoint;  { end points }
             next : LineList  { link to next one }
           END;
```

Regularized set operations between two polygons A and B are passed as a string of characters, they have the data structure

OpCode = PACKED ARRAY[1..6] OF CHAR; { A Op B }

The set of valid Opcode is: 'A I* B', 'A U* B', 'A -* B', and 'B -* A'.

3.3 2-D Algorithm

For a better understanding of the detailed algorithm which follows, an informal description of the algorithm is provided first:

1. {Input polygons} Read in the polygons A and B;
2. {Initialization} Initialize the segment lists XinB, Xtype1onB, Xtype2onB, and XoutB to nil;
3. {Membership classification function M'[A,B]} Separate into disjoint sub-segments each edge of Polygon A according to the classification function, then add each sub-segments to the appropriate list;
4. {Membership classification function M'[B,A]} Reverse the roles of polygons A and B and repeat steps 2 and 3;
5. {A Op B} Depending on which regularized set operation is desired, fetch edge segment lists according to Table I;
6. {Generate results} Merge the lists and traverse the list to generate polygons (there may be none, one, or several polygons). These polygons are the desired result of the operation.

In detail, the 2-D Polygon Modelling Algorithm is the following:

Step 0. {Declaration} Declare XinB, XinA, Xtype1onB, Xtype2onB,

Xtype1onA, Xtype2onA, XoutB, and XoutA as TYPE LineList and intPoints (intersection points) as TYPE aPolygon as defined in section 3.2;

Step 1. {Input polygons} Read in the two polygons A and B;

Step 2. {Initialization} Initialize the edge segment lists XinB, Xtype1onB, Xtype2onB, and XoutB to nil (Figure 10 illustrates the usage of these lists);

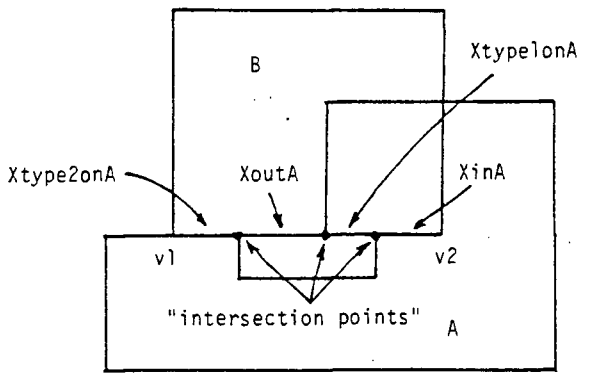

Figure 10. Segment and intersection point lists illustration.

Step 3. {Get polygon window} Get the minimal enclosing window of polygon B;

Step 4. {For each edge of A} Fetch an edge (v1,v2) of polygon A where v1, v2 are endpoints of the edge;

Step 5. {Initialize value} Initialize the intersection point list intPoints to nil;

Step 6. {Check clipping} If edge (v1,v2) is entirely outside the enclosing window of B, then add the edge to XoutB list, and go to step 19, else continue;

Step 7. {Add endpoints to intPoints} Add the endpoints v1 and

v2 to intPoints;

Step 8. {For each edge of B} Get an edge (c1,c2) of polygon B;

Step 9. If the two edges are parallel, then go to step 11, else continue;

Step 10. {Intersection case} If the two edges intersect at a point, then add that point to list intPoints; go to step 13;

Step 11. {Parallel case} If the two edges are not collinear, then go to step 13, else continue;

Step 12. {Collinear case} If the two edges have a common sub-segment, then test the segment for "on" type: if it is "Type 1", then add that segment to the list Xtype1onB, else add it to the list Xtype2onB (see section 2.4 for definitions of Xtype1onB and Xtype2onB and see Appendix B for a 3-D algorithm for deciding "Type 1" or "Type2" "on" cases); in either case, add the endpoints of the common segment to the list intPoints;

Step 13. Unless all edges for polygon B have been examined, fetch the next edge of polygon B, rename it to (c1,c2) and go to step 9; else continue;

Step 14. {Sort points} Sort all points of the intPoints list along edge (v1,v2) and eliminate any duplicate vertices; the resulting sorted list is assigned back to intPoints;

Step 15. {Fetch points which represent a segment} Fetch the first pair of points (t1,t2) from the list intPoints;

Step 16. {Find XinB and XoutB} If (t1,t2) is either in the Xtype1onB list or Xtype2onB list, then go to step 18, else continue;

Step 17. If the midpoint of (t1,t2) lies inside polygon B, then

add the segment (t1,t2) to XinB list, else add the segment to XoutB list (a point inclusion test is required here, see [6] [14] for an algorithm);

Step 18. If t2 was the last point in the intPoints list, then go to step 19, else fetch the next pair of points (e.g., the sequence t(1),t(2),..,t(n) will generate the pairs t(1)t(2),t(2)t(3),...,t(n-1)t(n)) and rename it (t1,t2), go to step 16;

Step 19. Unless all edges for polygon A have been examined, fetch the next edge of polygon A, rename it to (v1,v2) and go to step 5; else continue;

Step 20. {Reverse A and B and repeat entire algorithm} Reverse the roles of polygons A and B and repeat steps 2 to 19 with XinB, Xtype1onB, Xtype2onB, XoutB, and polygons A and B being replaced by XinA, Xtype1onA, Xtype2onA, XoutA, and polygons B and A respectively;

Step 21. {Fetch all line segments} Depending on the desired regularized set operation, select the line segment lists according to Table I;

Step 22. {Merging} Merge all segment lists to one list;

Step 23. {Generate result} Traverse the segments and generate closed polygons (there may be none, one, or several polygons), these polygons are the result of the desired operation.

3.4 Discussion and Result

The Cohen-Sutherland clipping algorithm [4] in step 6 was used for efficiency purposes [15]. The algorithm rejects all

edges that are not inside the window. In other words, these edges will not be intersecting with any edge of the polygon; hence, the extra calculation of comparing the edge with each edge of the polygon is avoided.

The point-in-polygon inclusion test [6] [14] in step 17 is important because it decides whether a point p is inside or outside the polygon.

The Pascal coding of a most parts of the 2-D polygon modelling algorithm can be found in Appendix A. Appendix C contains an example for which intermediate results for most steps are listed and the results of the four operations between two polygons A and B are generated as point lists. Figure 11 shows the results of the example in Appendix C.

The program has been run on several sets of data and the results obtained are correct. The output of an operation can serve as input for another operation. However, the algorithm can generate sets that are not in its domain, for example, polygons with holes. The result of the regularized union operation in the 2-D example of Appendix C is a polygon with a hole. Due to the representation scheme, only two polygons were generated as a result, without identifying the hole as such.

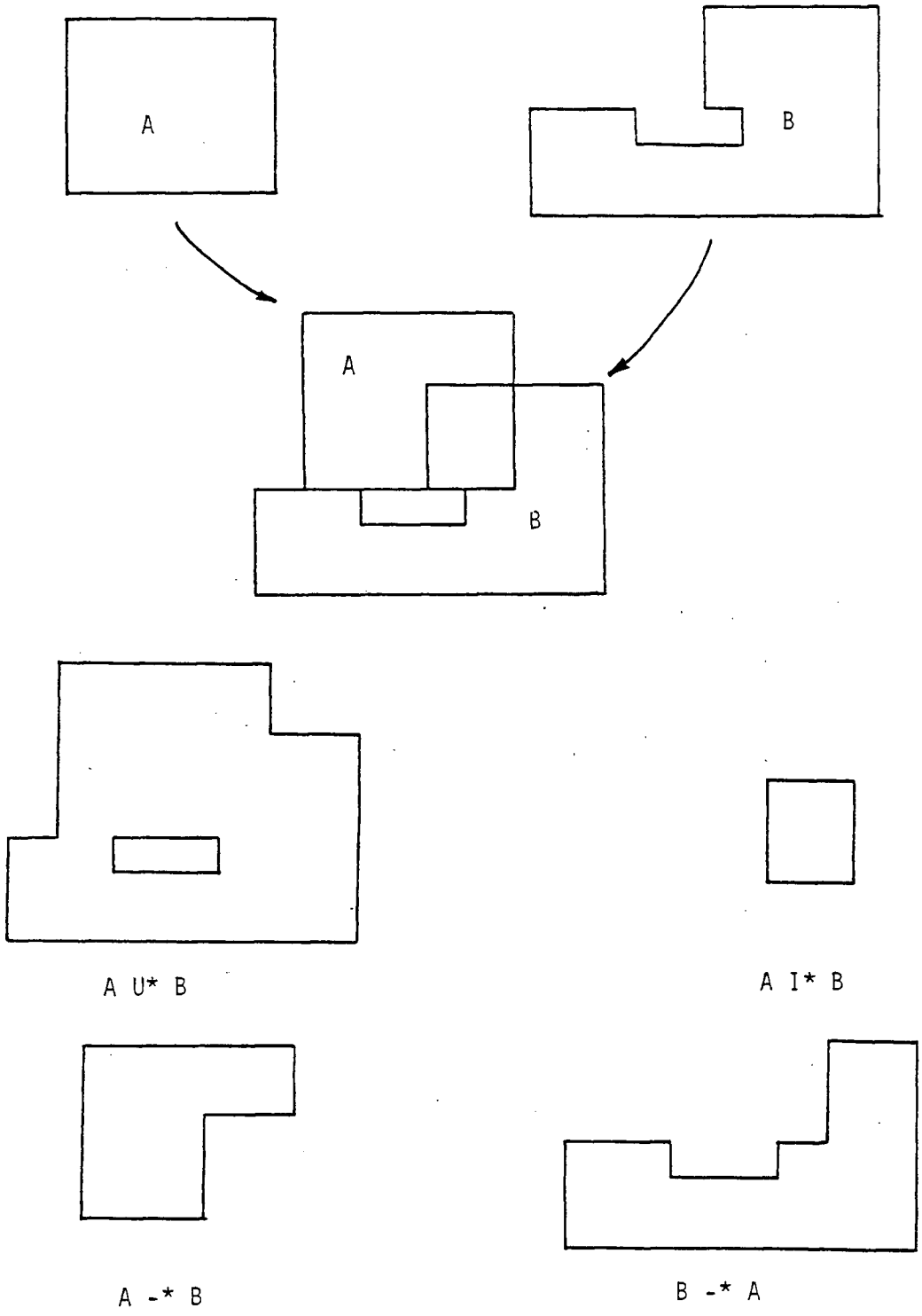

Figure 11. 2-D example.

CHAPTER 4

3-D POLYHEDRON MODELLING ALGORITHM

4.1 Domain of Algorithm

The domain of the Polyhedron Modelling Algorithm is the set of all 3-dimensional well-formed polyhedra whose faces are composed of polygons as defined in Chapter Two. However, polyhedra with holes in it are not in its domain.

Two polyhedra which touch at one point or which have only a common sub-segment of an edge are considered to be two distinct polyhedra.

4.2 Data Structure

The data structure of a polyhedron is illustrated by the following Pascal TYPEs :

```
XYZpoint = RECORD x, y, z : real END;  { (x,y,z) point }
VertexList = @Vertex;        { vertex list }
  Vertex = RECORD
             v : XYZpoint;
             next : VertexList
           END;
EdgeList = @Edges;           { edge list }
  Edges = RECORD
            e1, e2 : integer; { endpoints of edge }
            next : EdgeList
          END;
```

```
NumberList = @Numbers;      { number list }
  Numbers = RECORD
              n : integer;     { an edge }
              next : NumberList
            END;
FaceList = @Faces;          { face list }
  Faces = RECORD
            fnum : integer;    { number of edges }
            f : NumberList;    { set of edges }
            next : FaceList
          END;
Polyhedron = @Solid;        { a polyhedron }
  Solid = RECORD
            vnum,              { number of vertices }
            enum,              { number of edges    }
            snum : integer;    { number of faces    }
            vert : VertexList;    { vertex list }
            ed : EdgeList;        { edge list   }
            surf : FaceList       { face list   }
          END;
```

A polyhedron or solid is a record of six components. The number of vertices of the polyhedron is given by <u>vnum</u>, <u>enum</u> is the number of edges of the polyhedron, and <u>snum</u> represents the number of faces contained in the polyhedron.

<u>VertexList</u> contains all the vertices of the polyhedron. Each vertex is unique and is given by its (x,y,z) coordinates.

<u>EdgeList</u> is a pointer variable that points to a record of

three components. The first two components (e1,e2) represent an edge by the endpoints of that edge. The endpoints are integer numbers which tells the location of vertices in the VertexList. The last component, next, is a pointer to the next edge.

FaceList is a pointer variable that points to a record with three components. The first component, fnum, gives the number of edges of a face while the second component, f, is a linked NumberList which is a list of "fnum" integer numbers. Each number in the NumberList tells the location of the edge in the EdgeList. The last component, next, is a pointer to the next face.

A line is represented by its endpoints. A list of line segments is represented as follows:

```
LineList = @Vertices;            { a list of lines }
  Vertices = RECORD
                l1, l2 : XYZpoint; { end points }
                next : LineList    { link to next one }
             END;
```

EdgeSegment is a pointer variable that points to a record of five components. It contains the membership information of all edges of a polyhedron. The first component, enum, is the number assigned to this edge. Each edge is separated into disjoint sub-segments as defined by the membership classification function. Hence, the components, inl, onl, and, outl, contain the classified sub-segments of this edge. The last component, next, is a pointer to the next E_segments. The following TYPEs represent the data structure of EdgeSegment and

```
FaceSegment:
     EdgeSegment = @E_segments;      { a edge segment }
       E_segments = RECORD
                      enum : integer;  { edge number   }
                      inl : LineList;  { "in" segment  }
                      onl : LineList;  { "on" segment  }
                      outl : LineList; { "out" segment }
                      next : EdgeSegment
                    END;
     FaceSegment = @F_segments;        { a face segment }
       F_segments = RECORD
                      fnum : integer;     { face number }
                      flines : LineList; { a face }
                      next : FaceSegment
                    END;
```

FaceSegment is a pointer variable that points to a record of three components. It contains the membership information of all faces of a polyhedron. The first component, fnum, gives the number assigned to this face. The second component, flines, contains all the edges that constitute this face. The last component, next, is a pointer to the next F_segments.

```
     PointList = @Point;          { a list of points }
       Point = RECORD
                 num : integer;
                 p : VertexList
               END;
     Cut_Points_List = @CutPoints; { a list of cut-points }
```

```
CutPoints = RECORD
                c : PointList;
                next : Cut_Points_List
            END;
```

Cut Points List is a pointer variable that points to a record of two components and its data structure is represented above. The first component, c, is a pointer variable that points to a record of two fields. The first field, num, gives the edge number and the second field, p, is the set of all cut-points along the edge. The last component, next, is a pointer to the next list of CutPoints.

As in the 2-D data structure, the regularized set operations between two polyhedra are passed as a string of characters to a procedure; they have the data structure

aOpCode = PACKED ARRAY[1..6] OF CHAR;

The set of valid aOpCode is: "A I* B", "A U* B", "A -* B", and "B -* A".

4.3 3-D Algorithm

For a better understanding of the detailed algorithm which follows, an informal description of the algorithm is provided first:

1. {Input polyhedra} Read in the polyhedra A and B;
2. {Initialization} Initialize XinB, Xtype1onB, Xtype2onB, and XoutB to nil;
3. {Membership classification function M'[A,B]} Separate into disjoint sub-segments each face of polyhedron A

according to the classification function, then add each sub-segments to the appropriate list;

4. {Membership classification function M'[B,A]} Reverse the roles of polyhedra A and B and repeat steps 2 and 3;

5. {A Op B} Depending on which regularized set operation is desired, fetch face segment lists according to Table I;

6. {Generate results} Merge the lists and traverse the list to generate polyhedra (there may be none, one, or several polyhedra). These polyhedra are the desired result of the operation.

The detailed 3-D Polyhedron Modelling Algorithm is divided into five parts. Part I reads in the two polyhedra A and B. Part II classifies the self edges (a self edge is an edge of a polyhedron [10]) of the two polyhedra by using a line/solid classification. Part III creates all the cross edges (a cross edge is the line of intersection of two faces of two distinct polyhedra where the intersected line is not an self edge of either polyhedron [10]) between the two polyhedra by using a face/face intersection method. Part IV uses the results from Part II and Part III and classifies the faces of the two polyhedra. Part V generates polyhedra output depending on the desired regularized set operation. The detailed algorithm is the following:

Step 0. {Global declaration} Declare XinA, Xtype1onA, Xtype2onA, XoutA, XinB, Xtype1onB, Xtype2onB, XoutB, AFaceList, BFaceList, ACrossList, BCrossList as TYPE FaceSegment;

AEdgeList and BEdgeList as TYPE EdgeSegment; A_Edge_intPoints, B_Edge_intPoints as TYPE Cut_Points_List; and All_Cut_Lines as TYPE LineList;

PART I

INPUT POLYHEDRA

Step 1. {Input polyhedra} Read in the two polyhedra A and B;

PART II

SELF EDGE CLASSIFICATION

Step 2. {Global initialization} Initialize All_Cut_Lines to nil;

Step 3. {Initialization} Initialize B_Edge_intPoints, BFaceList, AEdgeList lists to nil;

Step 4. {Get polyhedron volume} Get the minimal enclosing volume of polyhedron B;

Step 5. {For each edge of A} Fetch an edge (v1,v2) of polyhedron A where v1, v2 are endpoints of each edge;

Step 6. {Assign edge number} Assign each edge a number according to the order it was fetched (that is, the first edge gets number 1, the second 2, and so on);

Step 7. {Initialize value} Initialize the intersection point list intPoints (TYPE PointList) to nil and set InList, OnList, and OutList (TYPE LineList) to nil;

Step 8. {Check clipping} If edge (v1,v2) is entirely outside

the enclosing volume of B, then add the edge to OutList, and go to step 26, else continue;

Step 9. {For each face of B} Get a face of polyhedron B;

Step 10. {Assign face number} Assign the face a number according to the order it was fetched (that is, the first face gets number 1, the second 2, and so on);

Step 11. {Initialize value} Initialize CutList of TYPE LineList to nil;

Step 12. If the edge is parallel to the face, then go to step 14, else continue;

Step 13. {Intersection case} If the edge cuts the face at a point, then add that point to list intPoints; go to step 17;

Step 14. {Parallel case} If the edge does not lie on the plane defined by the face, then go to step 17, else continue;

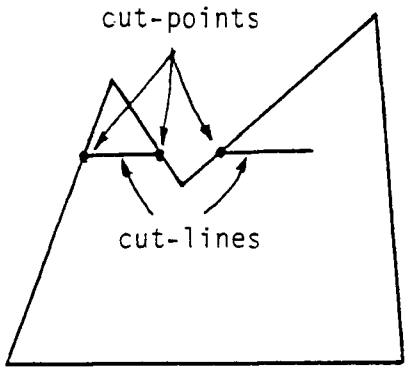

Figure 12. Cut-lines and cut-points on a face.

For the next step, the following terms are needed. A CUT-LINE is defined as a line that lies inside a face and a CUT-POINT is an endpoint of a cut-line and a point on the edge of the face as well. Figure 12 illustrates examples of cut-lines and cut-points. Note: the endpoints of a cut-line need not stop

at an edge of a face, it may stop at a point that is inside the face [5].

Step 15. {Add cut-lines} Determine the cut-lines of the edge, i.e., those parts of the edge that fall within the face. (This is similar to steps 2 to 19 of the 2-D polygon modelling algorithm in section 3.3 of finding XinB. Basically, the edge is tested against all edges of the face. First all intersection points are located, then the points are ordered along the edge and any duplicate points are eliminated, finally, the points are taken in pairs to form a segment and are tested whether the segment lies on the face by using a point inclusion test). Store all cut-lines in both OnList and CutList lists;

Step 16. {Add cut-points} Store all endpoints of cut-lines in intPoints;

Step 17. {Add face list} If there is a record with a face number that matches the current face number in the BFaceList, then merge the CutList with the existing cut-lines list, else create a new record (TYPE F_segments) of three components, fnum, flines, and next, and store the value of face number, CutList, and nil to fnum, flines, and next respectively; append the record to BFaceList;

Step 18. Unless all faces for polyhedron B have been examined, fetch the next face of polyhedron B and go to step 10; else continue;

Step 19. {Make copy} Make a copy of the list intPoints to intVertices;

Step 20. {Add endpoints} Add endpoints of edge (v1,v2) to

intVertices if they are not in that list;

Step 21. {Sort points} Sort all points of the intVertices list along edge (v1,v2) and eliminate any duplicate vertices; the resulting sorted list is assigned back to intVertices;

Step 22. {Fetch points which represent a segment} Fetch the first pair of points (t1,t2) from the list intVertices;

Step 23. {Find "in" and "out" type line segments} If (t1,t2) is not in the OnList, then continue, else go to step 25;

Step 24. If the midpoint of (t1,t2) lies inside polyhedron B, then add the segment (t1,t2) to InList, else add the segment (t1,t2) to OutList (a point inclusion test is required here, see [7] for an algorithm);

Step 25. If t2 was the last point in the intVertices list, then go to step 26, else fetch the next pair of points (e.g., the sequence t(1),t(2),..,t(n) will generate the pairs t(1)t(2),t(2)t(3),...,t(n-1)t(n)) and rename it (t1,t2), go to step 23;

Step 26. {Add edge list} Create a record (TYPE E_segments) and store the lists InList, OnList, and OutList and the associated edge number to this record. This record is then inserted at the end of the AEdgeList list;

Step 27. {Eliminate duplicate points} Eliminate any duplicate points of intPoints and the resulting list is assigned back to intPoints;

Step 28. Create a record and store intPoints and the current edge number to this record. This record is then inserted at the end of the B_Edge_intPoints list;

Step 29. {Append lists} Append InList, OnList, and OutList into All_Cut_Lines list;

Step 30. Unless all edges for polyhedron A have been examined, fetch the next edge of polyhedron A, rename it to (v1,v2) and go to step 6; else continue;

Step 31. {Reverse A and B and repeat entire algorithm} Reverse the roles of polyhedra A and B and repeat steps 3 to 30 with B_Edge_intPoints, BFaceList, AEdgeList, and polyhedra A and B being replaced by A_Edge_intPoints, AFaceList, BEdgeList, and polyhedra B and A respectively;

PART III

CROSS EDGE CLASSIFICATION

Step 32. {Initialize values} Initialize A_Cross_Edges and B_Cross_Edges (TYPE FaceSegment) to nil;

Step 33. {For each face of A} Get a face of polyhedron A;

Step 34. {Assign face number} Assign the face a number according to the order it was fetched (that is, the first face gets number 1, the second 2, and so on);

Step 35. {Get all cut-points of face} Initialize A_Face_cutPoints (TYPE VertexList) to nil;

Step 36. {For each edge of the face} Fetch an edge of the face;

Step 37. {Fetch cut-lines} Get the associated cut-points list of this edge from A_Edge_intPoints;

Step 38. {Make copy} Make a copy of the edge's cut-points list;

Step 39. {Merging} Merge this list with A_Face_cutPoints and

assign the value back to A_Face_cutPoints;

Step 40. Unless all edges for the current face from polyhedron A have been fetched, get the next edge of the face and go to step 37, else continue;

Step 41. {Eliminate duplicate points} Eliminate any duplicate points of A_Face_cutPoints, the resulting list is assigned back to A_Face_cutPoints;

Step 42. {For each face of B} Get a face of polyhedron B;

Step 43. {Assign face number} Assign the face a number according to the order it was fetched (that is, the first face gets number 1, the second 2, and so on);

Step 44. {Initialize value} Initialize CrossOnEdges (TYPE LineList) to nil;

Step 45. If the two faces are parallel, then go to step 59, else continue;

Step 46. {Line of intersection} Determine the line of intersection of the two planes containing the two faces;

Step 47. {Get all cut-points of face} repeat steps 35 to 41 with the list A_Face_cutPoints replaced by B_Face_cutPoints and using the face from polyhedron B in step 42 this time;

Step 48. {Filter all cut-points} Initialize OnLinePoints (TYPE VertexList) to nil;

Step 49. Fetch a vertice from A_Face_cutPoints;

Step 50. If the vertices is a point on the line of intersection, then add the vertice to OnLinePoints list;

Step 51. Unless all vertices of the A_Face_cutPoints list have been examined, fetch the next vertex and go to step 50, else

continue;

Step 52. {Filter all cut-points} Repeat steps 49 to 51 with A_Face_cutPoints replaced by B_Face_cutPoints (if the vertices that are on the line of intersection, continue to add to OnLinePoints list);

Step 53. If number of vertices in OnLinePoints is less than two, then go to step 59, else continue;

Step 54. {Sort points} Sort all points of the OnLinePoints list along the line of intersection and eliminate any duplicate vertices; the resulting sorted list is assigned back to OnLinePoints;

Step 55. {Fetch points which represent a segment} Fetch the first pair of points (t1,t2) from the list OnLinePoints;

Step 56. {Find cross edges} If (t1,t2) is neither in the CrossOnEdges list nor All_Cut_Lines list, then continue, else go to step 58;

Step 57. If the midpoint of (t1,t2) lies on the the current examined faces of polyhedra A and B, then add the segment (t1,t2) to CrossOnEdges;

Step 58. If t2 was the last point in the OnLinePoints list, then go to step 59, else fetch the next pair of points (e.g., the sequence t(1),t(2),..,t(n) will generate the pairs t(1)t(2),t(2)t(3),...,t(n-1)t(n)) and rename it (t1,t2), go to step 56;

Step 59. If CrossOnEdges is equal to nil, then go to step 62, else continue;

Step 60. {Store cross edges} If there is a record with a face

number that matches the current polyhedron B face's face number in the B_Cross_Edges, then merge the CrossOnEdges with the existing cut-lines list, else create a new record (TYPE F_segments) of three components, fnum, flines, and next, and store the value of face number, CrossOnEdges, and nil to fnum, flines, and next respectively; append the record to B_Cross_Edges;

Step 61. {Store cross edges} Repeat step 60 with B_Cross_Edges and polyhedron B's face number being replaced by A_Cross_Edges and polyhedron A's face number respectively;

Step 62. Unless all faces for polyhedron B have been examined, fetch the next face of polyhedron B and go to step 43; else continue;

Step 63. Unless all faces for polyhedron A have been examined, fetch the next face of polyhedron A and go to step 34; else continue;

PART IV

MEMBERSHIP CLASSIFICATION

Step 64. {Initialization} Initialize XinA, Xtype1onA, Xtype2onA, and XoutA to nil;

Step 65. {For each face of A} Get a face of polyhedron A;

Step 66. {Initialize values} Initialize InEdges, OnEdges, and OutEdges (TYPE LineList) to nil;

Step 67. {Fetch cut-lines} Fetch the cut-line list corresponding to this face from the AFaceList;

Step 68. {Make copy} Make a copy of the cut-line list;

Step 69. {Merge lists} Merge the copied list into OnEdges list;

Step 70. Repeat steps 67 to 69 with AFaceList being replaced by ACrossList (Note: there may not be any cross edges associated with this face, in this case, nil is returned as value);

Step 71. Rename the cut-line list from ACrossList to aList;

Step 72. {For each edge of the face} Fetch an edge of the current face;

Step 73. {Fetch cut-lines} Fetch the edge record corresponding to this edge from AEdgeList;

Step 74. {Make copies} Make copies of the segment lists inl, onl, and outl;

Step 75. {Merge lists} Merge the copy of inl into InEdges, the copy of onl into OnEdges, and the copy of outl into OutEdges respectively;

Step 76. Unless all the edges of the face have been fetched, get the next edge and go to step 74, else continue;

Step 77. {Generate holes} Traverse the cross edges list, aList, and generate closed polygons (there may be none, one, or several polygons), these polygons are then stacked in list HoleList (TYPE FaceSegment);

Step 78. {Unmarked segments} Unmark all InEdges, OnEdges, and OutEdges;

Step 79. {Generate XinA} If InEdges is equal to nil, then go to step 87, else fetch an unmarked edge from InEdges;

Step 80. Traverse the InEdges and OnEdges lists to generate a closed polygon, marking each edge that constitutes a part of

the polygon;

Step 81. If the HoleList is equal to nil, then go to step 85, else continue;

Step 82. Fetch a polygon from the HoleList;

Step 83. {Check possible holes} If the polygon from the HoleList is inside the generated polygon, then merge the edges lists together to form one polygon, and delete the polygon from the HoleList;

Step 84. Unless all the polygons for the HoleList have been examined, fetch the next polygon and go to step 83, else continue;

Step 85. Insert the (resulting) polygon to the end of XinA list;

Step 86. Unless all the InEdges edges have been marked, fetch another unmarked edge and go to step 80, else continue;

Step 87. {Generate XoutA} Repeat steps 79 to 86 with XinA and InEdges being replaced by XoutB and OutEdges respectively (Note: the OnEdges list may be partially marked);

Step 88. {Generate Xtype1onA and Xtype2onA} If OnEdges is either equal to nil or all have been marked, then go to step 93, else fetch an unmarked edge from OnEdges (inherited from above, may be partially marked);

Step 89. Traverse the OnEdges lists to generate a closed polygon, marking each edge that constitutes a part of the polygon;

Step 90. If one of the edges of the polygon is a subset of aList, then add the polygon to XinA list and go to step 92,

else continue;

Step 91. {Decide type} If the polygon (face segment) is "Type 1", then add the polygon to the list Xtype1onA, else add it to the list Xtype2onA (see section 2.4 for definitions of Xtype1onA and Xtype2onA and see Appendix B for an algorithm for deciding "Type1" or "Type2 " "on" cases);

Step 92. Unless all the edges from the OnEdges list have been marked, fetch the next unmarked edge and go to step 89, else continue;

Step 93. Unless all the faces of polyhedron A have been fetched, get the next face and go to step 66, else continue;

Step 94. Repeat steps 64 to 93 with XinA, Xtype1onA, Xtype2onA, XoutA, and polyhedra A and B being replaced by XinB, Xtype1onB, Xtype2onB, XoutB, and polyhedra B and A respectively;

PART V

GENERATING THE RESULT

Step 95. {Fetch all line segments} Depending on the desired regularized set operation, select the face segment lists according to Table I;

Step 96. {Merging} Merge all segment lists to one list;

Step 97. {Join face segments} For those faces that have a common edge and lie on the same plane, merge them into one face;

Step 98. {Join line segments} For those edge segments of a face that have the same space direction, join them together to one

line segment;

Step 99. {Generate result} Traverse the segments and generate closed polyhedra (there may be none, one, or several polyhedra), these polyhedra are the result of the desired operation.

4.4 Discussion and Result

Although there are many similarities between the 2-D algorithm and the 3-D algorithm, the 3-D version is more complicated.

As in 2-D, the Cohen-Sutherland clipping algorithm [4] is used again in the 3-D algorithm for efficiency purposes.

Kalay's [7] algorithm is used in determining the spatial containment of a point in a polyhedron. The algorithm, denoted "the projection method", is of complexity O(n). In general, this algorithm is an extension of the 2-D point-in-polygon inclusion algorithm, although it is more complicated.

In Part II, the self edges of the polyhedra are classified first, and these self edges are used to infer the cross edge classification in Part III. Requicha [10] claims that all vertices of all cross edges lie in self edges; he gives an informal proof for that claim. Based on this claim, all intersection points of the self edges were saved. Hence, recomputing the vertices was avoided in finding cross edges. The cross edges are obtained by using a face/face intersection comparison.

Similiar for the 2-D case, the program has been run on

several sets of data and by visual inspection, the results obtained are correct. The output of an operation can serve as input for another operation. However, the algorithm can generate sets that are not in its domain, for example, polyhedra with holes. Figure 13 shows a tested example with an operation that would generate a hole. Appendix D contains an example for which intermediate results for most steps are listed and the results of the four operations between two polyhedra A and B are illustrated in Figure 14.

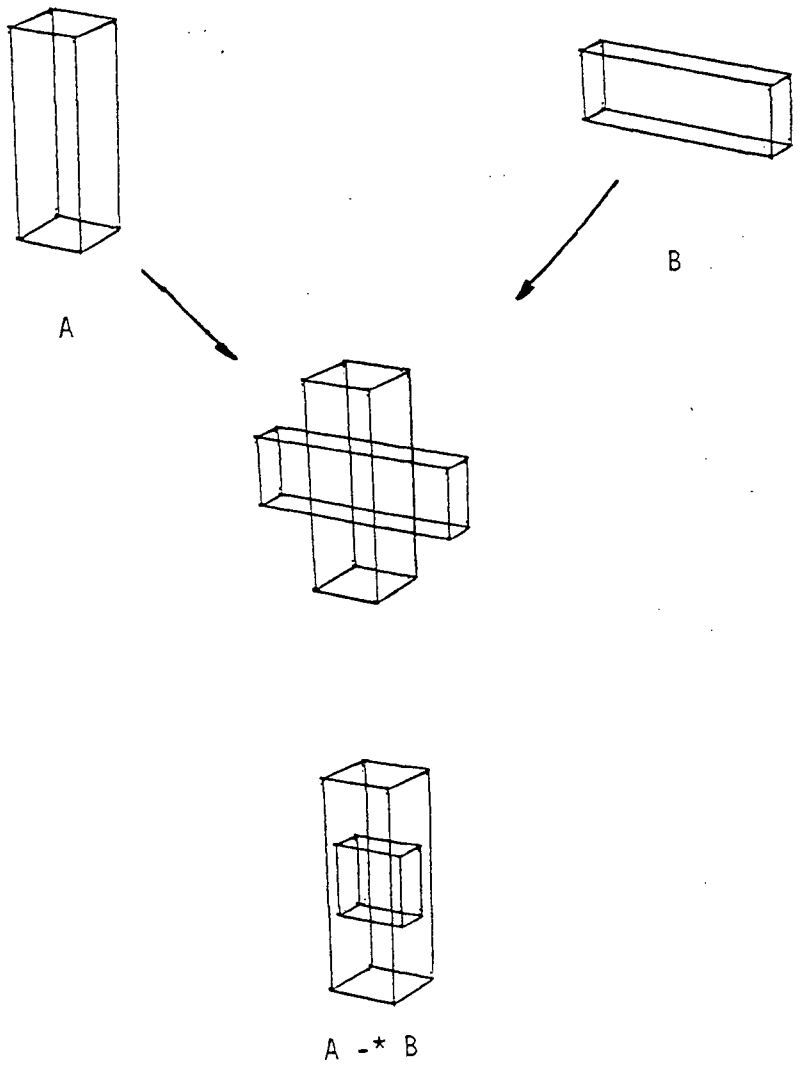

Figure 13. Example of an operation that generates an object with a hole in it.

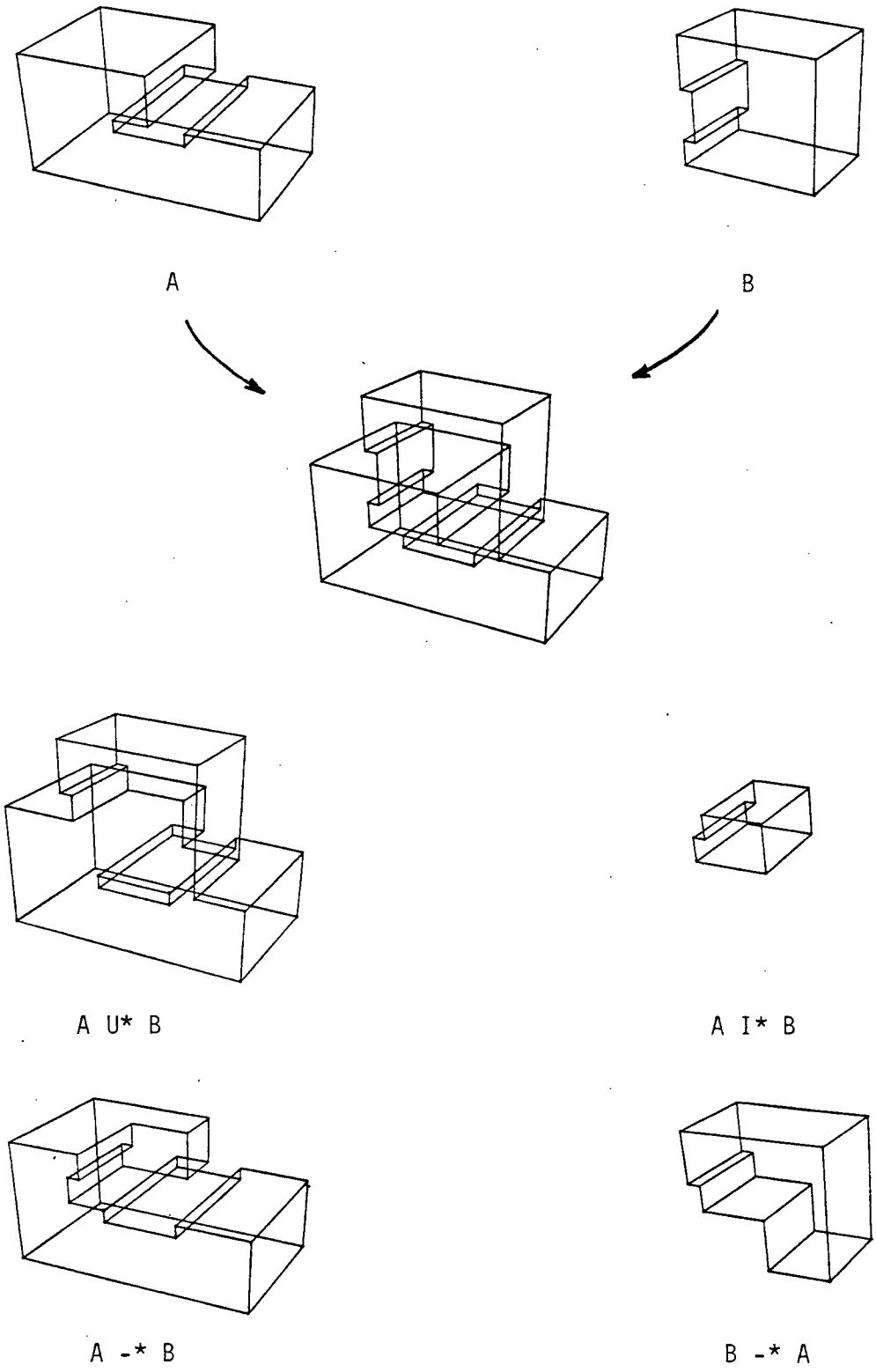

Figure 14. 3-D example.

CHAPTER 5

FUTURE WORK AND CONCLUSION

The domain of the polyhedron modelling algorithm presented in this thesis is not rich enough to cover most mechanical parts and tools in manufacturing. Hence, there is a need to extend the domain of this algorithm to incorporate holes and curved surfaces in objects.

Such an extension will be possible by using a better representation scheme for solids, while still retaining a large part of the presented algorithm.

Numerical accuracy is obviously poor in the current implementation due to memory limitations. For example, the program cannot distinguish between a line that is level and an inclined line that is almost level. It may be trivial in the current implementaion, but in industry, it is very important because precision is crucial in design and manufacturing.

During the design of the algorithms and their implementation, little consideration was given to their overall efficiency and complexity. Thus, these questions were not explicitly addressed in this thesis.

The thesis has shown that an algorithm for solid modelling, where the domain covers all polyhedra composed of convex and/or concave polygons, is implementable.

REFERENCES

[1] Boyse, J.W. and J.E. Gilchrist, "GMSolid: Interactive modeling for design and analysis of solids," IEEE Computer Graphics and Applications, Vol. 2, No. 2, Mar. 1982, pp. 27-40.

[2] Brown, C.M., "PADL-2: A technical summary," IEEE Computer Graphics and Applications, Vol. 2, No. 2, Mar. 1982, pp.69-84.

[3] Hillyard, R.C., "The build group of solid modelers," IEEE Computer Graphics and Applications, Vol. 2, No. 2, Mar. 1982, pp. 43-52.

[4] Foley, J.D. and A. Van Dam, "Fundamentals of Interactive Computer Graphics," Addison Wesley, Menlo Park, 1982.

[5] Franklin, W.R., "Efficient polyhedron intersection and union," Proc. Graphics Interface '82, pp. 73-80, Toronto, Canada, May 17-21, 1982.

[6] Hacker, R., "Certification of algorithm 112: Position of point relative to polygon," Comm. CAM, Vol. 5, Aug. 1962, pp. 606.

[7] Kalay, Y.E., "Determining the spatial containment of a point in general polyhedra," Computer Graphics & Image Processing, Vol. 19, No. 4, pp. 303-334, August 1982.

[8] Markowsky, G. and M.A. Wesley, "Fleshing out wireframes," IBM J. Research and Development, Vol. 24, No. 5, Sept. 1980, pp. 592-597.

[9] Meagher, D., "Geometric modelling using octree encoding," Computer Graphics & Image Processing, Vol. 19, No. 2, June 1982, pp. 129-147.

[10] Requicha, A.A.G., "Boolean operations on solids", Advanced Topics in Solid Modelling, Tutorial Notes, SIGGRAPH '83, Detroit, Michigan, July 26, 1983.

[11] Requicha, A.A.G., "Representations for rigid solids: theory, methods, and systems," ACM Computer Surveys, Vol. 12, No. 4, Dec. 1980, pp. 437-464.

[12] Requicha, A.A.G. and H.B. Voelcker, "Solid modelling: A historial summary and contemporary assessment", IEEE Computer Graphics & Applications, Vol. 2, No. 2, pp. 9-24, March 1982.

[13] Ross, R., "LIG6: Language for Interactive Graphics, User's Manual," Department of Electrical Engineering, The University of British Columbia, Vancouver, B.C., 1982.

[14] Shimrat, M., "Algorithm 112: Position of point relative to polygon," Comm. ACM, Vol. 5, Aug. 1962, pp. 434.

[15] Tilove, R.B., "Set membership classification: A unified approach to geometric intersection problems," IEEE Trans. Computers, Vol. C-29, No. 10, Oct. 1980, pp. 874-883.

APPENDIX A

2-D POLYGON MODELLING ALGORITHM

```
PROCEDURE Algorithm2D ( reference, candidate : aPolygon;
                        VAR In_list, Out_list,
                        On_list1, On_list2 : LineList );

{ Basic algorithm for polygon modelling }

  VAR
      v1, v2, pt1, pt2, c1, c2 : XYpoint;
      intPoints : aPolygon;
      Xmax, Xmin, Ymax, Ymin : Real;
      countCand, countRef : integer;

  BEGIN { Algorithm2D }

    GetPolygonWindow( reference, Xmax, Xmin, Ymax, Ymin );
    { initialize values }
    In_list := nil;            Out_list := nil;
    On_list1 := nil;           On_list2 := nil;

    { for each edge of candidate polygon do the following }
    FOR countCand := 1 TO candidate@.num DO BEGIN

       intPoints := nil;
       WRITELN; WRITELN( ' Polygon Line # ', countCand:3 );
       GetVertices( countCand, v1, v2, candidate );

       { check whether edge clips the window or not }
       IF Clipped( v1.x, v1.y, v2.x, v2.y,
                   Xmax, Xmin, Ymax, Ymin ) THEN BEGIN

          { store endpoints of edge (v1,v2) to intPoints }
          StorePoint( v1, intPoints );
          StorePoint( v2, intPoints );

          { Find intersection points.
            Compare the candidate edge with each
              reference edge }
          FOR countRef := 1 TO reference@.num DO BEGIN

            GetVertices( countRef, c1, c2, reference );
            { check whether the two edges are parallel }
            IF notParallel( c1, c2, v1, v2 ) THEN
               { locate intersection point }
               IF LineIntersect( c1, c2, v1, v2, pt1 )
                  THEN StorePoint( pt1, intPoints )
                  ELSE { do nothing }
```

```
        { check whether the two edges are collinear and
            have a common segment }
        ELSE IF LineOnLine( c1, c2, v1, v2, pt1, pt2 ) THEN
                { check on type }
                IF OnType( pt1, pt2, reference, candidate )
                   { Form on list }
                   THEN FormOnList( pt1, pt2, On_list1,
                                    intPoints )
                   ELSE FormOnList( pt1, pt2, On_list2,
                                    intPoints )

      END; { FOR countRef }

      { Form In_list and Out_list }
      FormIn_OutList( reference, intPoints, In_list,
                      Out_list, On_list1, On_list2 )
    END
  ELSE { edge totally outside window => out edge }
    FormOutList( v1, v2, Out_list ) { IF Clipped }

END { FOR countCand }

END; { Algorithm2D }
```

APPENDIX B

DECISION ALGORITHM "FINDONTYPE"

```
PROCEDURE FindOnType ( aSegment : LineList;
                       solid1, solid2 : Polyhedron;
                       VAR OnType1, OnType2 : FaceSegment );

{ Determine the OnType of a face segment }

  VAR
     p : XYZpoint;
     inside1, inside2 : boolean;


  BEGIN { FindOnType }

    { Find a point that is slightly above the surface
        of the face segment }
    OnePoint( p, aSegment );
    { Check if p is inside solid1 }
    IF Pt_inside_solid( p, solid1 ) THEN inside1 := true
                                    ELSE inside1 := false;
    { Check if p is inside solid2 }
    IF Pt_inside_solid( p, solid2 ) THEN inside2 := true
                                    ELSE inside2 := false;
    { determine type 1 or type 2 on cases }
    IF ( inside2 AND inside1 ) OR NOT ( inside2 OR inside1 )
       THEN BEGIN { same side => Type1 }
         StackResult( aSegment, OnType1 );
         WRITELN( ' Add to On1_Surface' )
       END
    ELSE BEGIN { opposite sides => Type2 }
         StackResult( aSegment, OnType2 );
         WRITELN( ' Add to On2_Surface' )
    END; { IF }
    { Print out aSegment }
    Printall( aSegment )

  END; { FindOnType }
```

APPENDIX C

2-D EXAMPLE

```
************************ ECHO INPUT ************************

@@@@@@@@@@@@@@@@@@@@@@@@ POLYGON  A @@@@@@@@@@@@@@@@@@@@@@@@
Vertices of polygon read in as follows:
      0.0        0.0
    10.00        0.0
    10.00       6.00
     5.00       6.00
     5.00       3.00
     6.00       3.00
     6.00       2.00
     3.00       2.00
     3.00       3.00
      0.0       3.00

@@@@@@@@@@@@@@@@@@@@@@@@ POLYGON  B @@@@@@@@@@@@@@@@@@@@@@@@
Vertices of polygon read in as follows:
     1.50       3.00
     7.50       3.00
     7.50       8.00
     1.50       8.00

**************** LINE/POLYGON CLASSIFICATION ****************
********************  Reference ( A ) ************************


Polygon Line #   1
  Added Out line  >>>    0.0    0.0            10.00    0.0

Polygon Line #   2
  Added Out line  >>>  10.00    0.0            10.00   6.00

Polygon Line #   3
  Added In  Line  >>>   5.00   6.00             7.50   6.00
  Added Out line  >>>   7.50   6.00            10.00   6.00

Polygon Line #   4
  Added In  Line  >>>   5.00   3.00             5.00   6.00

Polygon Line #   5
  Added On  list  >>>   5.00   3.00             6.00   3.00
```

```
Polygon Line #   6
  Added Out line  >>>   6.00  2.00            6.00  3.00

Polygon Line #   7
  Added Out line  >>>   6.00  2.00            3.00  2.00

Polygon Line #   8
  Added Out line  >>>   3.00  2.00            3.00  3.00

Polygon Line #   9
  Added On  list  >>>   1.50  3.00            3.00  3.00
  Added Out line  >>>    0.0  3.00            1.50  3.00

Polygon Line #  10
  Added Out line  >>>    0.0  3.00             0.0   0.0

**************** LINE/POLYGON CLASSIFICATION ****************
********************  Candidate ( B )  **********************


Polygon Line #   1
  Added On  list  >>>   5.00  3.00            6.00  3.00
  Added On  list  >>>   3.00  3.00            1.50  3.00
  Added Out line  >>>   3.00  3.00            5.00  3.00
  Added In  Line  >>>   6.00  3.00            7.50  3.00

Polygon Line #   2
  Added In  Line  >>>   7.50  3.00            7.50  6.00
  Added Out line  >>>   7.50  6.00            7.50  8.00

Polygon Line #   3
  Added Out line  >>>   7.50  8.00            1.50  8.00

Polygon Line #   4
  Added Out line  >>>   1.50  3.00            1.50  8.00

********************  OPERATION RESULT **********************

@@@@@@@@@@@@@@@@@@@@@@@@@@  A I* B  @@@@@@@@@@@@@@@@@@@@@@@@@@

Polygon with vertices as follows>>
 5.00  6.00
 5.00  3.00
 6.00  3.00
 7.50  3.00
 7.50  6.00

@@@@@@@@@@@@@@@@@@@@@@@@@@  A U* B  @@@@@@@@@@@@@@@@@@@@@@@@@@
```

```
Polygon with vertices as follows>>
  0.0   0.0
  0.0  3.00
 1.50  3.00
 1.50  8.00
 7.50  8.00
 7.50  6.00
10.00  6.00
10.00   0.0

Polygon with vertices as follows>>
 6.00  2.00
 3.00  2.00
 3.00  3.00
 5.00  3.00
 6.00  3.00

@@@@@@@@@@@@@@@@@@@@@@@@@@  A -* B  @@@@@@@@@@@@@@@@@@@@@@@@@@


Polygon with vertices as follows>>
  0.0   0.0
  0.0  3.00
 1.50  3.00
 3.00  3.00
 3.00  2.00
 6.00  2.00
 6.00  3.00
 7.50  3.00
 7.50  6.00
10.00  6.00
10.00   0.0

@@@@@@@@@@@@@@@@@@@@@@@@@@  B -* A  @@@@@@@@@@@@@@@@@@@@@@@@@@


Polygon with vertices as follows>>
 3.00  3.00
 1.50  3.00
 1.50  8.00
 7.50  8.00
 7.50  6.00
 5.00  6.00
 5.00  3.00
```

APPENDIX D

3-D EXAMPLE

```
***********************  ECHO INPUT  ***********************

################## SOLID Reference ( A ) ##################
 Vertices of solid read in as follows:
      0          0          0
     10.00       0          0
     10.00       5.00       0
      4.50       5.00       0
      4.50       3.00       0
      6.00       3.00       0
      6.00       2.50       0
      3.00       2.50       0
      3.00       3.00       0
      0          3.00       0
      0          0          6.00
     10.00       0          6.00
     10.00       5.00       6.00
      4.50       5.00       6.00
      4.50       3.00       6.00
      6.00       3.00       6.00
      6.00       2.50       6.00
      3.00       2.50       6.00
      3.00       3.00       6.00
      0          3.00       6.00
 Edges of solid read in as follows:
          1          2
          2          3
          3          4
          4          5
          5          6
          6          7
          7          8
          8          9
          9         10
         10          1
          1         11
          2         12
          3         13
          4         14
          5         15
          6         16
          7         17
          8         18
          9         19
         10         20
```

```
        11          12
        12          13
        13          14
        14          15
        15          16
        16          17
        17          18
        18          19
        19          20
        20          11
 Faces of solid read in as follows:
   1   2   3   4   5   6   7   8   9  10
  21  22  23  24  25  26  27  28  29  30
   1  11  21  12
   2  12  22  13
   3  13  23  14
   4  14  24  15
   5  15  25  16
   6  16  26  17
   7  17  27  18
   8  18  28  19
   9  19  29  20
  10  20  30  11

################## SOLID Candidate ( B ) ##################
 Vertices of solid read in as follows:
      2.00        3.00        0
      7.50        3.00        0
      7.50        4.00        0
      7.00        4.00        0
      7.00        6.00        0
      7.50        6.00        0
      7.50        8.00        0
      2.00        8.00        0
      2.00        3.00        4.00
      7.50        3.00        4.00
      7.50        4.00        4.00
      7.00        4.00        4.00
      7.00        6.00        4.00
      7.50        6.00        4.00
      7.50        8.00        4.00
      2.00        8.00        4.00
 Edges of solid read in as follows:
        1           2
        2           3
        3           4
        4           5
        5           6
        6           7
        7           8
        8           1
        9          10
       10          11
       11          12
```

12	13
13	14
14	15
15	16
16	9
1	9
2	10
3	11
4	12
5	13
6	14
7	15
8	16

Faces of solid read in as follows:

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
1	17	9	18				
2	18	10	19				
3	19	11	20				
4	20	12	21				
5	21	13	22				
6	22	14	23				
7	23	15	24				
8	24	16	17				

********** LINE/SOLID COMPARISON **********
********** Reference **********

Solid Line # 1							
Added Out Line	>	0	0	0	10.00	0	0
Solid Line # 2							
Added Out Line	>	10.00	0	0	10.00	5.00	0
Solid Line # 3							
Added On Line	>	7.00	5.00	0	4.50	5.00	0
Added Out Line	>	10.00	5.00	0	7.00	5.00	0
Solid Line # 4							
Added On Line	>	4.50	5.00	0	4.50	3.00	0
Solid Line # 5							
Added On Line	>	4.50	3.00	0	6.00	3.00	0
Solid Line # 6							
Added Out Line	>	6.00	3.00	0	6.00	2.50	0
Solid Line # 7							
Added Out Line	>	6.00	2.50	0	3.00	2.50	0
Solid Line # 8							
Added Out Line	>	3.00	2.50	0	3.00	3.00	0
Solid Line # 9							

Added On Line	>	3.00	3.00	0	2.00	3.00	0
Added Out Line	>	2.00	3.00	0	0	3.00	0
Solid Line # 10							
Added Out Line	>	0	3.00	0	0	0	0
Solid Line # 11							
Added Out Line	>	0	0	0	0	0	6.00
Solid Line # 12							
Added Out Line	>	10.00	0	0	10.00	0	6.00
Solid Line # 13							
Added Out Line	>	10.00	5.00	0	10.00	5.00	6.00
Solid Line # 14							
Added In Line	>	4.50	5.00	0	4.50	5.00	4.00
Added Out Line	>	4.50	5.00	4.00	4.50	5.00	6.00
Solid Line # 15							
Added On Line	>	4.50	3.00	0	4.50	3.00	4.00
Added Out Line	>	4.50	3.00	4.00	4.50	3.00	6.00
Solid Line # 16							
Added On Line	>	6.00	3.00	0	6.00	3.00	4.00
Added Out Line	>	6.00	3.00	4.00	6.00	3.00	6.00
Solid Line # 17							
Added Out Line	>	6.00	2.50	0	6.00	2.50	6.00
Solid Line # 18							
Added Out Line	>	3.00	2.50	0	3.00	2.50	6.00
Solid Line # 19							
Added On Line	>	3.00	3.00	0	3.00	3.00	4.00
Added Out Line	>	3.00	3.00	4.00	3.00	3.00	6.00
Solid Line # 20							
Added Out Line	>	0	3.00	0	0	3.00	6.00
Solid Line # 21							
Added Out Line	>	0	0	6.00	10.00	0	6.00
Solid Line # 22							
Added Out Line	>	10.00	0	6.00	10.00	5.00	6.00
Solid Line # 23							
Added Out Line	>	10.00	5.00	6.00	4.50	5.00	6.00
Solid Line # 24							
Added Out Line	>	4.50	5.00	6.00	4.50	3.00	6.00
Solid Line # 25							
Added Out Line	>	4.50	3.00	6.00	6.00	3.00	6.00

```
Solid Line #  26
  Added Out Line  >   6.00  3.00  6.00        6.00  2.50  6.00

Solid Line #  27
  Added Out Line  >   6.00  2.50  6.00        3.00  2.50  6.00

Solid Line #  28
  Added Out Line  >   3.00  2.50  6.00        3.00  3.00  6.00

Solid Line #  29
  Added Out Line  >   3.00  3.00  6.00        0     3.00  6.00

Solid Line #  30
  Added Out Line  >   0     3.00  6.00        0     0     6.00

******************  LINE/SOLID COMPARISON  *****************
************************   Candidate   ************************

Solid Line #   1
  Added On  Line  >   4.50  3.00  0           6.00  3.00  0
  Added On  Line  >   2.00  3.00  0           3.00  3.00  0
  Added On  Line  >   6.00  3.00  0           7.50  3.00  0
  Added Out Line  >   3.00  3.00  0           4.50  3.00  0

Solid Line #   2
  Added On  Line  >   7.50  3.00  0           7.50  4.00  0

Solid Line #   3
  Added On  Line  >   7.50  4.00  0           7.00  4.00  0

Solid Line #   4
  Added On  Line  >   7.00  4.00  0           7.00  5.00  0
  Added Out Line  >   7.00  5.00  0           7.00  6.00  0

Solid Line #   5
  Added Out Line  >   7.00  6.00  0           7.50  6.00  0

Solid Line #   6
  Added Out Line  >   7.50  6.00  0           7.50  8.00  0

Solid Line #   7
  Added Out Line  >   7.50  8.00  0           2.00  8.00  0

Solid Line #   8
  Added Out Line  >   2.00  8.00  0           2.00  3.00  0

Solid Line #   9
  Added On  Line  >   4.50  3.00  4.00        6.00  3.00  4.00
  Added On  Line  >   2.00  3.00  4.00        3.00  3.00  4.00
  Added Out Line  >   3.00  3.00  4.00        4.50  3.00  4.00
  Added In  Line  >   6.00  3.00  4.00        7.50  3.00  4.00

Solid Line #  10
```

```
  Added In  Line  >   7.50  3.00  4.00        7.50  4.00  4.00

Solid Line #  11
  Added In  Line  >   7.50  4.00  4.00        7.00  4.00  4.00

Solid Line #  12
  Added In  Line  >   7.00  4.00  4.00        7.00  5.00  4.00
  Added Out Line  >   7.00  5.00  4.00        7.00  6.00  4.00

Solid Line #  13
  Added Out Line  >   7.00  6.00  4.00        7.50  6.00  4.00

Solid Line #  14
  Added Out Line  >   7.50  6.00  4.00        7.50  8.00  4.00

Solid Line #  15
  Added Out Line  >   7.50  8.00  4.00        2.00  8.00  4.00

Solid Line #  16
  Added Out Line  >   2.00  8.00  4.00        2.00  3.00  4.00

Solid Line #  17
  Added On  Line  >   2.00  3.00  0           2.00  3.00  4.00

Solid Line #  18
  Added In  Line  >   7.50  3.00  0           7.50  3.00  4.00

Solid Line #  19
  Added In  Line  >   7.50  4.00  0           7.50  4.00  4.00

Solid Line #  20
  Added In  Line  >   7.00  4.00  0           7.00  4.00  4.00

Solid Line #  21
  Added Out Line  >   7.00  6.00  0           7.00  6.00  4.00

Solid Line #  22
  Added Out Line  >   7.50  6.00  0           7.50  6.00  4.00

Solid Line #  23
  Added Out Line  >   7.50  8.00  0           7.50  8.00  4.00

Solid Line #  24
  Added Out Line  >   2.00  8.00  0           2.00  8.00  4.00

********************  FIND CROSS EDGES  ********************

  Added On  Line  >   7.00  5.00  4.00        4.50  5.00  4.00
  Added On  Line  >   4.50  3.00  4.00        4.50  5.00  4.00
  Added On  Line  >   7.00  5.00  0           7.00  5.00  4.00

********************  GENERATING FACES  ********************
************************  Reference  ************************
```

```
Face #        1
 Add to Out_Surface
  0      0      0          10.00  0      0
 10.00   0      0          10.00  5.00   0
 10.00   5.00   0           7.00  5.00   0
  7.00   4.00   0           7.00  5.00   0
  7.50   4.00   0           7.00  4.00   0
  7.50   3.00   0           7.50  4.00   0
  6.00   3.00   0           7.50  3.00   0
  6.00   3.00   0           6.00  2.50   0
  6.00   2.50   0           3.00  2.50   0
  3.00   2.50   0           3.00  3.00   0
  3.00   3.00   0           2.00  3.00   0
  2.00   3.00   0           0     3.00   0
  0      3.00   0           0     0      0
 Add to On1_Surface
  7.00   5.00   0           4.50  5.00   0
  4.50   5.00   0           4.50  3.00   0
  4.50   3.00   0           6.00  3.00   0
  6.00   3.00   0           7.50  3.00   0
  7.50   3.00   0           7.50  4.00   0
  7.50   4.00   0           7.00  4.00   0
  7.00   4.00   0           7.00  5.00   0

Face #        2
 Add to Out_Surface
  0      0      6.00       10.00  0      6.00
 10.00   0      6.00       10.00  5.00   6.00
 10.00   5.00   6.00        4.50  5.00   6.00
  4.50   5.00   6.00        4.50  3.00   6.00
  4.50   3.00   6.00        6.00  3.00   6.00
  6.00   3.00   6.00        6.00  2.50   6.00
  6.00   2.50   6.00        3.00  2.50   6.00
  3.00   2.50   6.00        3.00  3.00   6.00
  3.00   3.00   6.00        0     3.00   6.00
  0      3.00   6.00        0     0      6.00

Face #        3
 Add to Out_Surface
  0      0      0          10.00  0      0
 10.00   0      0          10.00  0      6.00
  0      0      6.00       10.00  0      6.00
  0      0      0           0     0      6.00

Face #        4
 Add to Out_Surface
 10.00   0      0          10.00  5.00   0
 10.00   5.00   0          10.00  5.00   6.00
 10.00   0      6.00       10.00  5.00   6.00
 10.00   0      0          10.00  0      6.00

Face #        5
 Add to  In_Surface
  4.50   5.00   0           4.50  5.00   4.00
```

```
  7.00   5.00   4.00          4.50   5.00   4.00
  7.00   5.00   0             7.00   5.00   4.00
  7.00   5.00   0             4.50   5.00   0
 Add to Out_Surface
 10.00   5.00   0             7.00   5.00   0
  7.00   5.00   0             7.00   5.00   4.00
  7.00   5.00   4.00          4.50   5.00   4.00
  4.50   5.00   4.00          4.50   5.00   6.00
 10.00   5.00   6.00          4.50   5.00   6.00
 10.00   5.00   0            10.00   5.00   6.00

Face #        6
 Add to  In_Surface
  4.50   5.00   0             4.50   5.00   4.00
  4.50   3.00   4.00          4.50   5.00   4.00
  4.50   3.00   0             4.50   3.00   4.00
  4.50   5.00   0             4.50   3.00   0
 Add to Out_Surface
  4.50   5.00   4.00          4.50   5.00   6.00
  4.50   5.00   6.00          4.50   3.00   6.00
  4.50   3.00   4.00          4.50   3.00   6.00
  4.50   3.00   4.00          4.50   5.00   4.00

Face #        7
 Add to Out_Surface
  4.50   3.00   4.00          4.50   3.00   6.00
  4.50   3.00   6.00          6.00   3.00   6.00
  6.00   3.00   4.00          6.00   3.00   6.00
  4.50   3.00   4.00          6.00   3.00   4.00
 Add to On1_Surface
  4.50   3.00   0             6.00   3.00   0
  6.00   3.00   0             6.00   3.00   4.00
  4.50   3.00   4.00          6.00   3.00   4.00
  4.50   3.00   0             4.50   3.00   4.00

Face #        8
 Add to Out_Surface
  6.00   3.00   0             6.00   2.50   0
  6.00   2.50   0             6.00   2.50   6.00
  6.00   3.00   6.00          6.00   2.50   6.00
  6.00   3.00   4.00          6.00   3.00   6.00
  6.00   3.00   0             6.00   3.00   4.00

Face #        9
 Add to Out_Surface
  6.00   2.50   0             3.00   2.50   0
  3.00   2.50   0             3.00   2.50   6.00
  6.00   2.50   6.00          3.00   2.50   6.00
  6.00   2.50   0             6.00   2.50   6.00

Face #       10
 Add to Out_Surface
  3.00   2.50   0             3.00   3.00   0
  3.00   3.00   0             3.00   3.00   4.00
```

```
  3.00  3.00  4.00        3.00  3.00  6.00
  3.00  2.50  6.00        3.00  3.00  6.00
  3.00  2.50  0           3.00  2.50  6.00

Face #       11
 Add to Out_Surface
  2.00  3.00  0           0     3.00  0
  0     3.00  0           0     3.00  6.00
  3.00  3.00  6.00        0     3.00  6.00
  3.00  3.00  4.00        3.00  3.00  6.00
  2.00  3.00  4.00        3.00  3.00  4.00
  2.00  3.00  0           2.00  3.00  4.00
 Add to On2_Surface
  3.00  3.00  0           2.00  3.00  0
  2.00  3.00  0           2.00  3.00  4.00
  2.00  3.00  4.00        3.00  3.00  4.00
  3.00  3.00  0           3.00  3.00  4.00

Face #       12
 Add to Out_Surface
  0     3.00  0           0     0     0
  0     0     0           0     0     6.00
  0     3.00  6.00        0     0     6.00
  0     3.00  0           0     3.00  6.00

********************  GENERATING FACES  ********************
************************   Candidate   ************************

Face #        1
 Add to Out_Surface
  3.00  3.00  0           4.50  3.00  0
  4.50  5.00  0           4.50  3.00  0
  7.00  5.00  0           4.50  5.00  0
  7.00  5.00  0           7.00  6.00  0
  7.00  6.00  0           7.50  6.00  0
  7.50  6.00  0           7.50  8.00  0
  7.50  8.00  0           2.00  8.00  0
  2.00  8.00  0           2.00  3.00  0
  2.00  3.00  0           3.00  3.00  0
 Add to On1_Surface
  4.50  3.00  0           6.00  3.00  0
  6.00  3.00  0           7.50  3.00  0
  7.50  3.00  0           7.50  4.00  0
  7.50  4.00  0           7.00  4.00  0
  7.00  4.00  0           7.00  5.00  0
  7.00  5.00  0           4.50  5.00  0
  4.50  5.00  0           4.50  3.00  0

Face #        2
 Add to  In_Surface
  6.00  3.00  4.00        7.50  3.00  4.00
  7.50  3.00  4.00        7.50  4.00  4.00
  7.50  4.00  4.00        7.00  4.00  4.00
  7.00  4.00  4.00        7.00  5.00  4.00
```

```
  7.00  5.00  4.00        4.50  5.00  4.00
  4.50  3.00  4.00        4.50  5.00  4.00
  4.50  3.00  4.00        6.00  3.00  4.00
 Add to Out_Surface
  3.00  3.00  4.00        4.50  3.00  4.00
  4.50  3.00  4.00        4.50  5.00  4.00
  7.00  5.00  4.00        4.50  5.00  4.00
  7.00  5.00  4.00        7.00  6.00  4.00
  7.00  6.00  4.00        7.50  6.00  4.00
  7.50  6.00  4.00        7.50  8.00  4.00
  7.50  8.00  4.00        2.00  8.00  4.00
  2.00  8.00  4.00        2.00  3.00  4.00
  2.00  3.00  4.00        3.00  3.00  4.00

Face #        3
 Add to  In_Surface
  6.00  3.00  4.00        7.50  3.00  4.00
  7.50  3.00  0           7.50  3.00  4.00
  6.00  3.00  0           7.50  3.00  0
  6.00  3.00  0           6.00  3.00  4.00
 Add to Out_Surface
  3.00  3.00  0           4.50  3.00  0
  4.50  3.00  0           4.50  3.00  4.00
  3.00  3.00  4.00        4.50  3.00  4.00
  3.00  3.00  0           3.00  3.00  4.00
 Add to On1_Surface
  4.50  3.00  0           6.00  3.00  0
  6.00  3.00  0           6.00  3.00  4.00
  4.50  3.00  4.00        6.00  3.00  4.00
  4.50  3.00  0           4.50  3.00  4.00
 Add to On2_Surface
  2.00  3.00  0           3.00  3.00  0
  3.00  3.00  0           3.00  3.00  4.00
  2.00  3.00  4.00        3.00  3.00  4.00
  2.00  3.00  0           2.00  3.00  4.00

Face #        4
 Add to  In_Surface
  7.50  3.00  0           7.50  3.00  4.00
  7.50  3.00  4.00        7.50  4.00  4.00
  7.50  4.00  0           7.50  4.00  4.00
  7.50  3.00  0           7.50  4.00  0

Face #        5
 Add to  In_Surface
  7.50  4.00  0           7.50  4.00  4.00
  7.50  4.00  4.00        7.00  4.00  4.00
  7.00  4.00  0           7.00  4.00  4.00
  7.50  4.00  0           7.00  4.00  0

Face #        6
 Add to  In_Surface
  7.00  4.00  0           7.00  4.00  4.00
  7.00  4.00  4.00        7.00  5.00  4.00
```

```
  7.00   5.00   0            7.00   5.00   4.00
  7.00   4.00   0            7.00   5.00   0
 Add to Out_Surface
  7.00   5.00   0            7.00   6.00   0
  7.00   6.00   0            7.00   6.00   4.00
  7.00   5.00   4.00         7.00   6.00   4.00
  7.00   5.00   0            7.00   5.00   4.00

Face #        7
 Add to Out_Surface
  7.00   6.00   0            7.50   6.00   0
  7.50   6.00   0            7.50   6.00   4.00
  7.00   6.00   4.00         7.50   6.00   4.00
  7.00   6.00   0            7.00   6.00   4.00

Face #        8
 Add to Out_Surface
  7.50   6.00   0            7.50   8.00   0
  7.50   8.00   0            7.50   8.00   4.00
  7.50   6.00   4.00         7.50   8.00   4.00
  7.50   6.00   0            7.50   6.00   4.00

Face #        9
 Add to Out_Surface
  7.50   8.00   0            2.00   8.00   0
  2.00   8.00   0            2.00   8.00   4.00
  7.50   8.00   4.00         2.00   8.00   4.00
  7.50   8.00   0            7.50   8.00   4.00

Face #       10
 Add to Out_Surface
  2.00   8.00   0            2.00   3.00   0
  2.00   3.00   0            2.00   3.00   4.00
  2.00   8.00   4.00         2.00   3.00   4.00
  2.00   8.00   0            2.00   8.00   4.00

********************  OPERATION RESULT  ********************

#########################  A I* B  #########################

   12
    4.50     3.00     4.00
    7.50     3.00     4.00
    7.50     4.00     4.00
    7.00     4.00     4.00
    7.00     5.00     4.00
    4.50     5.00     4.00
    7.50     3.00     0
    4.50     3.00     0
    7.50     4.00     0
    7.00     4.00     0
    7.00     5.00     0
    4.50     5.00     0
   18
```

```
  1     2
  2     3
  3     4
  4     5
  5     6
  1     6
  7     2
  8     7
  8     1
  9     3
  7     9
 10     4
  9    10
 11     5
 10    11
 12     6
 11    12
 12     8
 8
  6
 1    2    3    4    5    6
  4
 1    7    8    9
  4
 7    2   10   11
  4
10    3   12   13
  4
12    4   14   15
  4
16    5   14   17
  4
16    6    9   18
  6
17   18    8   11   13   15

#########################  A U* B  #########################

 33
  3.00     3.00     0
  6.00     3.00     0
  7.00     5.00     0
  7.00     6.00     0
  7.50     6.00     0
  7.50     8.00     0
  2.00     8.00     0
  2.00     3.00     0
  0        0        0
 10.00     0        0
 10.00     5.00     0
  6.00     2.50     0
  3.00     2.50     0
  0        3.00     0
  2.00     3.00     4.00
```

4.50	3.00	4.00
4.50	5.00	4.00
7.00	5.00	4.00
7.00	6.00	4.00
7.50	6.00	4.00
7.50	8.00	4.00
2.00	8.00	4.00
3.00	3.00	4.00
6.00	3.00	6.00
4.50	3.00	6.00
0	0	6.00
10.00	0	6.00
10.00	5.00	6.00
4.50	5.00	6.00
6.00	2.50	6.00
3.00	2.50	6.00
3.00	3.00	6.00
0	3.00	6.00

52

1	2
3	4
4	5
5	6
6	7
7	8
9	10
10	11
11	3
2	12
12	13
13	1
8	14
14	9
15	16
16	17
18	17
18	19
19	20
20	21
21	22
22	15
23	16
1	23
2	24
16	25
25	24
4	19
3	18
5	20
6	21
7	22
8	15
26	27
27	28

28 29
29 25
24 30
30 31
31 32
32 33
33 26
10 27
9 26
11 28
17 29
12 30
13 31
1 32
14 33
23 32
15 23
18
14
1 2 3 4 5 6 7 8 9 10 11 12 13 14
8
15 16 17 18 19 20 21 22
6
1 23 24 25 26 27
4
2 28 18 29
4
3 30 19 28
4
4 31 20 30
4
5 32 21 31
4
6 33 22 32
10
34 35 36 37 27 38 39 40 41 42
4
7 43 34 44
4
8 45 35 43
6
9 29 17 46 36 45
4
46 37 26 16
4
10 47 38 25
4
11 48 39 47
4
12 49 40 48
6
13 50 41 51 52 33
4
14 44 42 50

```
#########################  A -* B  #########################

   16
    2.00      3.00      0
    4.50      3.00      0
    4.50      5.00      0
    7.00      5.00      0
    7.00      6.00      0
    7.50      6.00      0
    7.50      8.00      0
    2.00      8.00      0
    2.00      3.00      4.00
    4.50      3.00      4.00
    4.50      5.00      4.00
    7.00      5.00      4.00
    7.00      6.00      4.00
    7.50      6.00      4.00
    7.50      8.00      4.00
    2.00      8.00      4.00
   24
    1     2
    3     2
    4     3
    4     5
    5     6
    6     7
    7     8
    8     1
    9    10
   10    11
   12    11
   12    13
   13    14
   14    15
   15    16
   16     9
    2    10
    1     9
    5    13
    4    12
    6    14
    7    15
    8    16
    3    11
  10
    8
   1    2    3    4    5    6    7    8
    8
   9   10   11   12   13   14   15   16
    4
   1   17    9   18
    4
   4   19   12   20
```

```
  4
 5  21   13   19
  4
 6  22   14   21
  4
 7  23   15   22
  4
 8  18   16   23
  4
24  11   20    3
  4
24  10   17    2

##########################  B -* A  ##########################

 29
  0        0        0
 10.00     0        0
 10.00     5.00     0
  7.00     5.00     0
  7.00     4.00     0
  7.50     4.00     0
  7.50     3.00     0
  6.00     3.00     0
  6.00     2.50     0
  3.00     2.50     0
  3.00     3.00     0
  0        3.00     0
  0        0        6.00
 10.00     0        6.00
 10.00     5.00     6.00
  4.50     5.00     6.00
  4.50     3.00     6.00
  6.00     3.00     6.00
  6.00     2.50     6.00
  3.00     2.50     6.00
  3.00     3.00     6.00
  0        3.00     6.00
  7.00     5.00     4.00
  4.50     5.00     4.00
  4.50     3.00     4.00
  6.00     3.00     4.00
  7.50     3.00     4.00
  7.50     4.00     4.00
  7.00     4.00     4.00
 46
  1     2
  2     3
  3     4
  5     4
  6     5
  7     6
  8     7
  8     9
```

```
  9    10
 10    11
 11    12
 12     1
 13    14
 14    15
 15    16
 16    17
 17    18
 18    19
 19    20
 20    21
 21    22
 22    13
  2    14
  1    13
  3    15
  4    23
 23    24
 24    16
 25    17
 25    24
 26    18
 25    26
  9    19
  8    18
 10    20
 11    21
 12    22
 25    27
 27    28
 28    29
 29    23
 26    27
  7    27
  8    26
  6    28
  5    29
17
 12
 1    2    3    4    5    6    7    8    9   10   11   12
 10
13   14   15   16   17   18   19   20   21   22
  4
 1   23   13   24
  4
 2   25   14   23
  6
 3   26   27   28   15   25
  4
28   16   29   30
  4
29   17   31   32
  4
```

8	33	18	34		
4					
9	35	19	33		
4					
10	36	20	35		
4					
11	37	21	36		
4					
12	24	22	37		
6					
38	39	40	41	27	30
4					
42	43	7	44		
4					
43	39	45	6		
4					
45	40	46	5		
4					
46	41	26	4		