UMIACS–TR–88–48
CS–TR–2057

July, 1988

A General Method for Performing Set Operations on Polyhedra†

George Vaněček Jr.
Department of Computer Science

Dana Nau
Institute for Advanced Computer Studies
Computer Science Department
University of Maryland
College Park, MD 20742

ABSTRACT

Three fundamental binary set operations are of interest in solid modeling: union, intersection and subtraction. This paper describes an efficient algorithm for set operations on pairs of polyhedra modeled using boundary representation; the polyhedra need not be bounded by manifolds. Given the boundaries of the two solids being operated upon, the algorithm uses boundary classification to separate the two boundaries into four classification sets each. This is done using an efficient boundary classification method based on non–regular decomposition of object space. The algorithm then constructs the boundary of the desired object form the appropriate classification sets.

† The work has been supported in part by an NSF Presidential Young Investigator Award to Dana Nau with matching funds provided by General Motors Research Laboratories and Texas Instruments, and NSF Grant NSFD CDR–85–00108 to the University of Maryland Systems Research Center.

1. Introduction

It is well known that when set-theoretic operations such as union, intersection, and subtraction are applied to two valid n-dimensional solids, the result is not necessarily a valid n-dimensional solid. For example, if two squares touch on one side, their set-theoretic intersection is a single line segment which is not a valid 2-dimensional solid. For solid modeling, one is normally interested not in the direct result of the set operation, but rather in the *regularized* result of the set operation [RV77,RV85], i.e., that part of the result that is a valid n-dimensional solid.

If the two solids are represented using boundary representations, a common approach to computing a regularized set operation on these two solids is to make use of boundary classification. Boundary classification was formalized by Tilove [Til77,Til80], and was elaborated on by Putnam and Subrahmaniam [PS86], and Mäntylä [Man84b]. The intent of boundary classification is to put each face of each solid into one of several different classification sets, depending on whether a face is inside, outside, or on the boundary of the other solid. Once this has been done, the boundary of the resulting solid can be constructed based on the classification sets.

During the boundary classification process, some faces may touch or intersect the other solid only partially, and these faces cannot be put into the classification sets directly. Such faces are first split into several subfaces such that each subface lies completely inside, outside, or on the other solid—and then these subfaces are put into the classification sets.

When solids are represented by boundary representations (BReps), algorithms for set operations on these solids often are rather inefficient. During the computation of set operations, a typical approach for boundary classification is to check each edge of one solid against every face of the other solid in order to find out where they intersect. This requires time $O(n^2)$, typically with a rather large constant factor. As a result, the time required to perform set operations may become prohibitive if the solids are complex.

Another problem that arises in performing set operations on BReps is how to handle non-manifold objects. A 2-manifold M is a space such that each point on M has an open neighborhood topologically equivalent to an open disk of E^2 [Man84a]. Although most physical solids are bounded by 2-manifolds, the result of a regularized set operation on two solids bounded by 2-manifolds need

not necessarily be bounded by a 2-manifold [Req77]. A general algorithm that implements the set operations on solids must therefore be capable of representing and manipulating solids that are not strictly bound by 2-manifolds—and this causes problems for BRep modelers that use data structures such as winged-edge representations [Bau72].

This paper presents a new algorithm for set operations on polyhedra represented using boundary representations. This algorithm provides better average-case performance than the quadratic cost of the conventional approach to boundary classification. This is accomplished by using a divide-and-conquer approach to decompose the boundaries recursively into fragments that can be classified quickly. The decomposition is done by separating space into regions using splitting planes lying along the faces of the solids. This means that the edges of one solid are never checked against the other solid (as is done in the conventional approach).

In addition to efficiency, there are several other appealing characteristics of this algorithm. It operates directly on the boundary representations (without requiring conversion to and from other representations), it does not have to deal with numerous special cases as do many existing algorithms, it works for both 2- and 3-dimensional solids, the boundaries of the solids do not have to be manifolds, and the basic concept is easy to describe.

This paper discusses the operation of the algorithm on 3-dimensional polyhedra. As discussed in [VN87b], the algorithm becomes simpler if applied to the problem of intersecting collections of 2-dimensional polygons.

Section 2, gives a brief overview of related work that has been done on decomposition techniques. Section 3 discusses the details of the algorithm. Section 4 discusses implementation considerations, and Section 6 provides concluding remarks.

2. Related Work

The idea of performing space decomposition by use of a plane is not new. Quadtrees and octrees use regular decomposition of space to represent rectilinear solids. Extended octrees provide an efficient and exact representation of polyhedra. Isabel Navazo in her Ph.D. thesis [Nav86], and Ayala [ABN85] showed how to perform set operations on extended octrees. Non-regular decomposition was first used by Fuchs in his presentation of Binary Space Partitioning

Trees (BSP) to represent polyhedra for the use in a hidden surface algorithm [FKN80]. The use of BSP trees was recently extended by Thibault and Naylor to allow for the conversion from CSG to BSP to BRep, and to allow for the computation of set operations between a BSP and a CSG primitive [TN87]. Of the previous work, Naylor's approach is the most similar to ours, because we both perform a similar type of decomposition.

The main differences between our work and the approaches discussed above are as follows:

1. Each of these approaches proceeds by building elaborate data structures to represent localized regions of space, and then using these data structures as the underlying representations for the objects being manipulated (for example, Thibault and Naylor map BReps into BSP trees and do their manipulations on these trees). In contrast, our algorithm deals purely with boundary representations without converting to and from an alternative representation.

2. Each of the approaches makes certain simplifying assumptions about the nature of the objects being represented. For example, all of the approaches assume manifold boundaries, and many of them assume that the faces of the solids are convex. In contrast, we handle arbitrary polyhedral objects.

3. The Algorithm

3.1. Definitions, and Overview of the Algorithm

A plane $\mathcal{P}$ is represented by a four-tuple (A, B, C, D). (A, B, C) is the normal vector for $\mathcal{P}$, and the distance from $\mathcal{P}$ to the origin is $|D|$. A point (x, y, z) is in $\mathcal{P}$ if $Ax + By + Cz + D = 0$.

$$\mathcal{P}_< = \{(x, y, z) \mid Ax + By + Cz + D < 0\}$$

is the half-space *below* $\mathcal{P}$, and

$$\mathcal{P}_\geq = \{(x, y, z) \mid Ax + By + Cz + D \geq 0\}$$

is the half-space *on or above* $\mathcal{P}$.

A solid can be modeled as a subset of E^3 that is bounded, closed, regular, and semianalytic [Req80,RV83] (one consequence of this is that every solid has

a finite volume). The mathematical properties of such solids are discussed by Requicha [RV77]. If S is a polyhedral solid, then bS is the set of faces that bound S. If f is a face of S, then $\mathcal{P}_f$ is the plane (A,B,C,D) containing f. To give the orientation of f with respect to S, the signs of A, B, C, and D are chosen so that f's normal vector $\overline{N}_f = (A,B,C)$ points outward from S. f^{-1} is the face occupying the same space as f, but having the normal vector $(-A,-B,-C)$ pointing in the opposite direction. If F is a set of faces, then $F^{-1} = \{f^{-1} \mid f \in F\}$.

A *region* is any non-empty intersection of zero or more open or closed half-spaces. The *fragment* F^S of a solid S within a region R is

$$F^S = \{f \cap^* R \mid f \in F^S\}.$$

The use of regularized intersection here is to handle the case where a boundary of R is open rather than closed. As a consequence of this, the interior of each face in F^S is in R, but the borders of each face in F^S need not necessarily be in R.

Regions are useful conceptually for describing the algorithms in this paper, but none of the algorithms ever deal with R explicitly. Thus, the problem of how to compute the above regularized intersection never arises.

A face f of S is *homogeneous* with respect to solid T if one or more of the following classification relationships holds:

1. f **IN** T; i.e., the interior of f lies in the interior of T.
2. f **OUT** T; i.e., the interior of f is outside of T.
3. f **WITH** T; i.e., f lies on the boundary of T, and both S and T are on the same side of the boundary.
4. f **ANTI** T; i.e., f lies on the boundary of T, and S and T are on opposite sides of the boundary.

Not every face of S is necessarily homogeneous with respect to T, and non-homogeneous faces cannot be classified directly. Instead, they must be split into two or more subfaces, each of which is homogeneous. Once this has been done, the classification relationships for the subfaces can be determined.

A fragment F^S of S is homogeneous with respect to T if all of its faces satisfy the same classification relationships with respect to T. In order to classify

the faces of S and T, a decomposition procedure will be used that implicitly partitions the space into regions, and tries to classify the fragments in those regions. If a fragment is found to be homogeneous, then it may be classified directly (see Sections 3.5 and 3.6). Otherwise, the region containing the fragment is decomposed into subregions in hopes that the subfragments in the subregions will be homogeneous. The algorithm for regularized binary set operations is as follows:

1. Use the decomposition procedure described in Section 3.2 to divide bS and bT into sets $b_T(S)$ and $b_S(T)$ of homogeneous faces and classify them. This computes the following eight *classification sets*:

$$S_{\mathbf{IN}}T = \{f \in b_T(S) \mid f \textbf{ IN } T\}; \tag{3.1}$$
$$S_{\mathbf{OUT}}T = \{f \in b_T(S) \mid f \textbf{ OUT } T\}; \tag{3.2}$$
$$S_{\mathbf{WITH}}T = \{f \in b_T(S) \mid f \textbf{ WITH } T\}; \tag{3.3}$$
$$S_{\mathbf{ANTI}}T = \{f \in b_T(S) \mid f \textbf{ ANTI } T\}; \tag{3.4}$$
$$T_{\mathbf{IN}}S = \{f \in b_S(T) \mid f \textbf{ IN } S\}; \tag{3.5}$$
$$T_{\mathbf{OUT}}S = \{f \in b_S(T) \mid f \textbf{ OUT } S\}; \tag{3.6}$$
$$T_{\mathbf{WITH}}S = \{f \in b_S(T) \mid f \textbf{ WITH } S\}; \tag{3.7}$$
$$T_{\mathbf{ANTI}}S = \{f \in b_S(T) \mid f \textbf{ ANTI } S\}. \tag{3.8}$$

2. The classification sets computed in Step 1 allow for the boundary of the new solid to be constructed by incrementally copying and then gluing together the faces of the appropriate classification sets. Depending on which operation is to be computed (union, intersection, or difference), compute it using one of the following equations:

$$B = b(S \cup^* T) = S_{\mathbf{OUT}}T \cup T_{\mathbf{OUT}}S \cup S_{\mathbf{WITH}}T, \tag{3.9}$$
$$B = b(S \cap^* T) = S_{\mathbf{IN}}T \cup T_{\mathbf{IN}}S \cup T_{\mathbf{WITH}}S, \tag{3.10}$$
$$B = b(S -^* T) = S_{\mathbf{OUT}}T \cup (T_{\mathbf{IN}}S)^{-1} \cup S_{\mathbf{ANTI}}T, \tag{3.11}$$
$$B = b(T -^* S) = T_{\mathbf{OUT}}S \cup (S_{\mathbf{IN}}T)^{-1} \cup T_{\mathbf{ANTI}}S. \tag{3.12}$$

3. Due to the decomposition process, $b_T(S)$ and $b_S(T)$ probably consist of non-maximal faces; i.e., they contain faces which are coplanar and adjacent, and which therefore could be combined. Thus, when the boundary

B is constructed using one of equations (3.9) through (3.12), it will probably also contain non-maximal faces. Such boundaries are non-unique, and their use in further set operations could proliferate the number of non-maximal faces to a point where the boundaries could consist of many more faces than needed. Therefore, at this point we perform a face merging operation to form maximal faces. This is done in two steps. The first step is to merge adjacent distinct coplanar faces by removing from B the edges which lie between them. Once this has been done, the merged faces may result in collinear and adjacent edges. The second step merges these edges by removing from B all vertices that have exactly two adjacent collinear edges.

4. The resulting boundary B may consist of one or more disjoint components which need to be separated, and some of these components might be non-manifold. If a component is not a manifold, it may consist of several subcomponents whose interiors are disjoint; these subcomponents also need to be separated. We check B for the existence of such situations, and separate B into single-component boundaries. (This task must be done here rather than during Step 2, since the classification sets computed during Step 2 do not provide any information about the relationships among the desired faces and the structure of the resulting solid(s).)

As an example of Step 3 above, Figure 3.1a shows a solid with non-maximal faces. Figure 3.1b shows the solid after face- and edge-merging. For visualization, Figure 3.2 shows the solid with hidden faces removed.

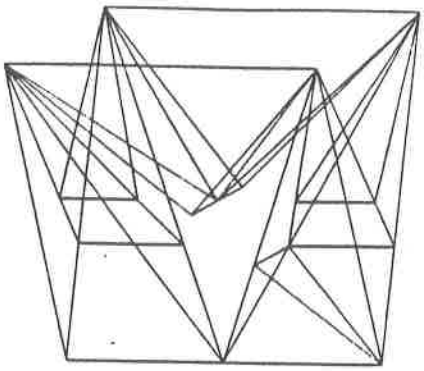

Figure 3.1a

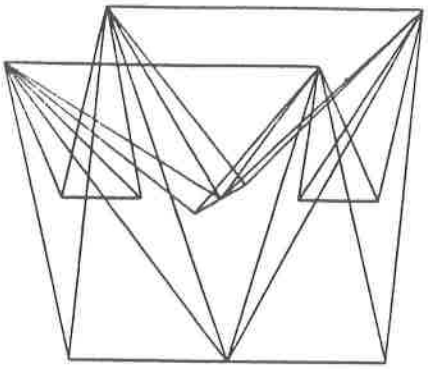

Figure 3.1b

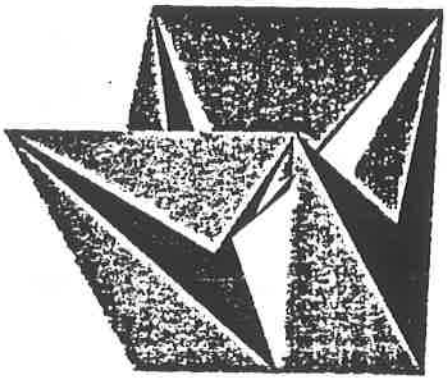

Figure 3.2: Solid of Figure 3.1 with hidden faces removed.

In Step 4 above, the resulting boundary needs to be separated into connected components. However, looking for connected components alone will not fully separate the boundary if the boundary is not a 2-manifold. As an example, Figure 3.3 shows a non-manifold solid consisting of a rectangular peg inserted in a cavity of a larger solid. Complete separation can be achieved by propagating connectivity among all strongly connected faces as shown in Figure 3.4.

3.2. The Decomposition Procedure

This section describes the decomposition procedure mentioned in Step 1 of the set operation algorithm. This procedure is used to divide two fragments into

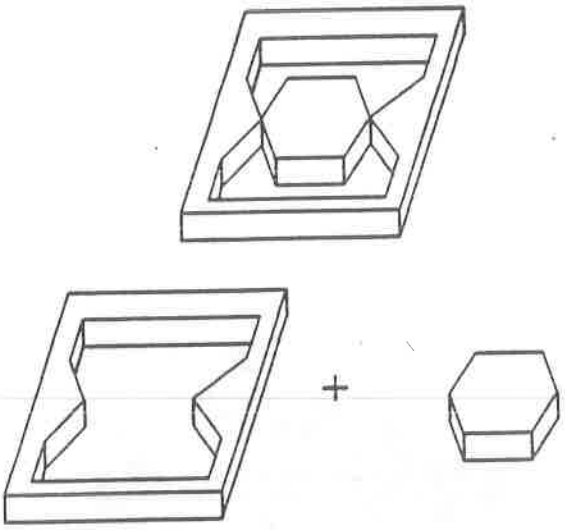

Figure 3.3: Separating a non-manifold into two components.

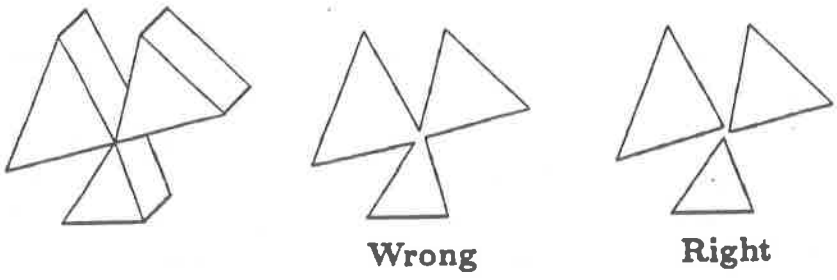

Figure 3.4: Proper propagation of face connectivity.

sets of homogeneous faces.

Let F^S and F^T be the fragments of S and T in some region R. (initially, R will be the entire universe, and F^S and F^T will be bS and bT, respectively). Unless it is known that F^S and F^T are homogeneous, R (and thus F^S and F^T) must be decomposed in order to simplify the problem. This is done by splitting R into two subregions. The procedure never makes any explicit use of R, but since F^S and F^T are the only entities of interest in R, splitting F^S and F^T correspond implicitly to splitting R. F^S and F^T are split by intersecting them with the half-spaces defined by a plane called the *splitting plane*. The splitting plane is either a plane that contains one of the faces of F^S and F^T, or else a plane perpendicular to one of these faces. How the splitting plane is selected is discussed in Section 3.3.

When a region R is split by a splitting plane $\mathcal{P}$, the resulting subregions are $R_{\geq} = R \cap \mathcal{P}_{\geq}$ (the portion of region R on or above $\mathcal{P}$), and $R_{<} = R \cap \mathcal{P}_{<}$ (the portion of region R below $\mathcal{P}$). The corresponding fragments of S and T are

$$F^S_{\geq} = \{f \cap^* \mathcal{P}_{\geq} \mid f \in F^S\}; \tag{3.13}$$

$$F^S_{\geq} = \{f \cap^* \mathcal{P}_{\geq} \mid f \in F^S\}; \tag{3.14}$$

$$F^T_{<} = \{f \cap^* \mathcal{P}_{<} \mid f \in F^T\}; \tag{3.15}$$

$$F^T_{<} = \{f \cap^* \mathcal{P}_{<} \mid f \in F^T\}. \tag{3.16}$$

How these fragments are computed is described in Section 3.4.

The recursive decomposition procedure appears below.

1. From the arguments F^S and F^T, choose a splitting plane $\mathcal{P}$ using the splitting-plane selection procedure described in Section 3.3.

2. Using $\mathcal{P}$, split F^S into subfragments $F^S_{<}$ and $F^S_{\geq}$, and F^T into subfragments $F^T_{<}$ and $F^T_{\geq}$, using the fragment splitting procedure described in Section 3.4. As an example, Figure 3.5 shows a 6-faced polyhedron split into two fragments with 5 faces each.

3. (Since the following steps are similar for both the fragments on or above $\mathcal{P}$ and the fragments below $\mathcal{P}$, only the steps for the fragments on or above $\mathcal{P}$ are presented.) Either classify the faces of $F^S_{\geq}$ and $F^T_{\geq}$ directly, or decompose them recursively. One of the following steps accomplishes this:

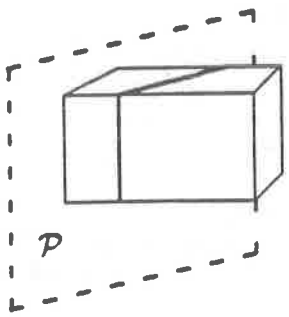

Figure 3.5: Splitting a solid by a plane $\mathcal{P}$.

- If $F_{\geq}^{S}$ and $F_{\geq}^{T}$ contain one face each, and if the two faces share a common border, classify both faces using the classification procedure of Section 3.5.
- If $F_{\geq}^{S}$ is empty, classify the faces of the fragment $F_{\geq}^{T}$ directly as members of either the $T_{\mathrm{IN}}S$ or the $T_{\mathrm{OUT}}S$ classification sets, using the classification procedure of Section 3.6.
- If $F_{\geq}^{T}$ is empty, classify the faces of the fragment $F_{\geq}^{S}$ directly as members of either the $S_{\mathrm{IN}}T$ or the $S_{\mathrm{OUT}}T$ classification sets, using the classification procedure of Section 3.6.
- Otherwise, if both $F_{\geq}^{S}$ and $F_{\geq}^{T}$ are non-empty, invoke the decomposition procedure on them recursively.

3.3. Selecting a Splitting Plane

This section describes the procedure for selecting the splitting plane chosen in Step 1 of the decomposition procedure (Section 3.2). The splitting plane is chosen by first selecting a face from the valid faces of the two fragments to be decomposed, and then by choosing a plane based on the relationship between the face and the region that contains it.

The splitting plane is usually taken to be a plane that contains a face of one of the fragments. Using such splitting planes is sufficient when the fragments cleanly intersect, i.e., they do not have overlapping faces. However, if the

fragments do have overlapping faces, then this is not sufficient to decompose the fragments into homogeneous subfragments. Since the overlapping faces are coplanar, the problem becomes one of decomposing a 2-dimensional region. The overlapping faces can be decomposed by splitting them along any line that is collinear with an edge of one of the faces. In our implementation of the algorithm, this is done by choosing a splitting plane that is perpendicular to the overlapping faces.

For the divide-and-conquer approach to work correctly and efficiently, it is necessary for the decomposed regions to be independent of each other, and for them to be reasonably balanced in size. Thus, not every face or edge is appropriate to determine the splitting plane. To pick a proper face, it is necessary to know the relationship between a given face f and the region R that contains the face. For 3-dimensional space, there are five face-region relationships:

f **In**$_3$ R holds if the interior of f lies completely inside R and is not coplanar with any of the planes comprising the the boundary of R.

f **With**$_3$ R holds if f is coplanar with one of the planes bounding R and f and the plane have the same the normal vectors (i.e., f and the plane face the same direction). For example, see Figure 3.6a.

f **Anti**$_3$ R holds if f is coplanar with one of the planes bordering R and the normal vectors for f and the plane are different (i.e., f and the plane face in opposite directions). For example, see Figure 3.6b.

f **In**$_2$ R holds if both f **With**$_3$ R and f **Anti**$_3$ R. This indicates that the face is imbedded in a planar (that is, 2-dimensional) region. Thus, all faces of this fragment are coplanar.

f **WA**$_2$ R holds if f **In**$_2$ R and the border of R is exactly equal to the border of f (in which case f is the only face in the fragment that contains it).

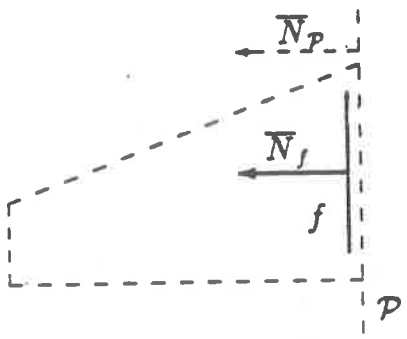

Figure 6a: $\mathbf{With}_3$

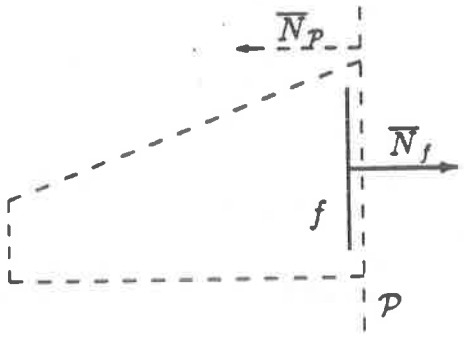

Figure 6b: $\mathbf{Anti}_3$

Suppose F^S and F^T contain faces satisfying various of the above relationships. Then, in selecting a face to use in defining the splitting plane, it is necessary to give preference to faces satisfying the first possible relation in the sequence

$$(\mathbf{In}_3, \mathbf{With}_3, \mathbf{Anti}_3, \mathbf{In}_2).$$

A face satisfying the $\mathbf{WA}_2$ relation will never appear as a candidate, since the fragment containing the face would already be classified in Step 3 of the decomposition procedure and would not require further decomposition.

Once a face f has been selected, the selection procedure chooses a splitting plane $\mathcal{P}$ as follows:

1. If $f\mathbf{In}_3 R$ or $f\mathbf{Anti}_3 R$, then $\mathcal{P}$ is the plane (A, B, C, D) of f.
2. If $f\mathbf{With}_3 R$, then $\mathcal{P}$ is the plane $(-A, -B, -C, -D)$ of f^{-1}.
3. If $f\mathbf{In}_2 R$, then $\mathcal{P}$ is a plane which is perpendicular to the plane of f. $\mathcal{P}$ is chosen to contain an edge of f that lies completely within the region R (at least one such edge must exist, since otherwise f would satisfy $f\mathbf{WA}_2 R$, whence the fragment containing f would not require splitting).

3.4. Splitting a Fragment

The fragment splitting procedure uses the splitting plane $\mathcal{P}$ provided by the selection procedure to split a fragment F into two subfragments. One of the subfragments, $F_{\geq}$, lies on or above the splitting plane, and the other subfragment, $F_{<}$, lies below. Any face coplanar with the splitting plane belongs to $F_{\geq}$. The fragment splitting procedure has the following steps:

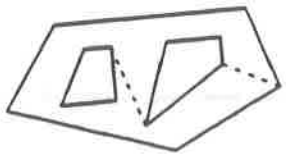

Figure 3.7: Representing a face with holes.

1. Initially, set $F_{\geq} = F_{<} = \emptyset$.
2. For each face f in the fragment F,
   - If f lies on or above $\mathcal{P}$, then put f into $F_{\geq}$.
   - If f lies below $\mathcal{P}$, then put f into $F_{<}$.
   - If f crosses the splitting plane $\mathcal{P}$, use the face-splitting procedure of Section 3.4.1 to split f into two or more faces, each of which lies either on or above $\mathcal{P}$, or below $\mathcal{P}$. Place each new face in either $F_{\geq}$ or $F_{<}$, as appropriate.
3. Return the new subfragments $F_{<}$ and $F_{\geq}$.

A face f is determined to lie above, below, or across $\mathcal{P}$ by noting the position of each edge of f with respect to $\mathcal{P}$. This is done by noting the positions of the vertices of f. If $p = (x, y, z)$ is a vertex of f and (A, B, C, D) is the 4-tuple representing $\mathcal{P}$, then p is below, above, or on $\mathcal{P}$ if the distance

$$d(p) = Ax + By + Cz + D$$

from p to $\mathcal{P}$ is less than, equal to, or greater than 0, respectively. If all vertices of f are below $\mathcal{P}$, then f is below $\mathcal{P}$; if all vertices of f are on or above $\mathcal{P}$, then f is on or above $\mathcal{P}$; and otherwise, f crosses $\mathcal{P}$.

When considering a multiply connected face (i.e. a face with holes), the hole can be represented as a loop [BHS80]. Splitting each face can be greatly simplified if the holes do not need to be maintained separately, but can be incorporated into the outer loop of the face using imaginary edges [YT85]; this is the approach we use in the implementation of the algorithm. Figure 3.7 shows a face with two holes, each connected by a single edge. This representation allows the face splitting procedure to ignore holes and to handle splitting faces with holes identically to splitting faces without holes.

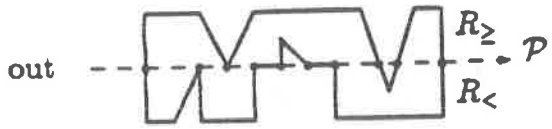

Figure 3.8: Splitting a face.

3.4.1. Splitting a Face

The problem of determining the subfaces lying on only one side of the splitting plane has been widely investigated in computer graphics as the polygon clipping problem [SH74]; however, such discussions are oriented towards graphic display rather than solid modeling, so that maintaining a consistent solid is unnecessary. For the purpose of set operations, cutting polyhedra by splitting planes has been investigated by Brotz [Bro76] and Mäntylä [Man86]—but these papers assume that the polyhedra have manifold boundaries. This section describes our procedure for determining what subfaces are generated when a face is split by the splitting plane, with the purpose of elucidating some of the details necessary for face-splitting that have not been addressed elsewhere.

The face-splitting procedure uses the splitting plane $\mathcal{P}$ to cut a face f into two or more faces. This involves the following steps:

1. Split all edges of f that cross the splitting plane $\mathcal{P}$. This results in adding new vertices to f; all of these new vertices lie in $\mathcal{P}$.

2. let V be a list of all vertices of f which lie in $\mathcal{P}$, sorted in order along the line of intersection between $\mathcal{P}$ and the plane of f.

3. The point-set intersection between $\mathcal{P}$ and f consists of one or more line segments. Some of these line segments may already correspond to edges of f; for the others, we create edges bounded by the vertices in V.

4. The previous step splits f into two or more subfaces. Of these subfaces, add to $F_{\geq}$ the ones lying on or above $\mathcal{P}$, and add the rest to $F_{<}$.

In Step 2, the set V is constructed by tracing around the edges of f in a counterclockwise order. To insert the new edges in Step 3, it is first necessary

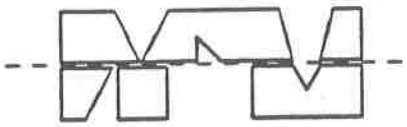

Figure 3.9: Result of Splitting a face.

in Step 2 to sort V along the line of intersection between $\mathcal{P}$ and the plane of f. This line may be represented by the parametric equation

$$p(v) = p(v_1) + t \cdot \overline{w}, \qquad (*)$$

where v_1 is an arbitrary vertex in V, and

$$\overline{w} = \overline{N}_f \times \overline{N}_{\mathcal{P}}.$$

The goal is to order the vertices $v \in V$ according to the value $t(v)$, where $t(v)$ is obtained by solving Eq. $(*)$ for t. Fortunately, computing $t(v)$ can be avoided by sorting on the coordinate of v that corresponds to the coordinate of $\overline{w}$ having the greatest magnitude. For example, if $\overline{w} = (c_1, c_2, c_3)$ and $\max\{c_1, c_2, c_3\} = c_2$, then the 2nd coordinate of v.

Sorting V is not by itself sufficient to determine where to insert the new edges in Step 3, because not every adjacent pair of vertices in V will correspond to an edge. For example, Figure 3.8 shows a face in which V contains ten vertices. Of these ten vertices, the fifth and the sixth vertex from the left should not have a new edge created between them. Tracing along the vector $\overline{w}$, we are originally outside f. Every time we encounter a vertex v in V, we can determine whether we have entered or exited f by checking to see whether the vertices close to v along the boundary of f are on opposite sides of $\mathcal{P}$. If we are inside f, then an edge should be added.

Once the new edges have been inserted, this splits f into a number of new faces. For example, Figure 3.9 shows the result of splitting the face shown in Figure 3.8: the edge loop for the original face has been broken up into six edge loops by adding five new edges, and each new edge loop defines a new face.

3.5. WITH/ANTI Classification Procedure

In Step 3 of the decomposition procedure, suppose the two fragments to be classified contain one face each, and that these faces share a common border. Then these faces are homogeneous, and can be classified without further decomposition.

Let F^S and F^T be the fragments, and let f_S and f_T, respectively, be the faces contained in these fragments. There two possibilities: either the normal vectors of f_S and f_T are opposite, or else they are equal. In the first case, the solids S and T touch (i.e., they are on opposite sides of the faces, as in Figure 3.10a). In this case, classifying f_S and f_T consists of adding them to the classification sets $S_{\mathrm{ANTI}}T$ and $T_{\mathrm{ANTI}}S$, respectively. In the second case, the solids S and T overlap (i.e., they are on the same side of the faces, as in Figure 3.10b). In this case, classifying f_S and f_T consists of adding them to $S_{\mathrm{WITH}}T$ and $T_{\mathrm{WITH}}S$, respectively.

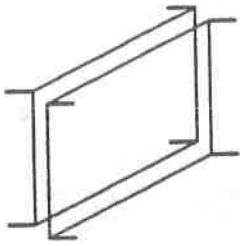

Figure 10a: Solids touch.

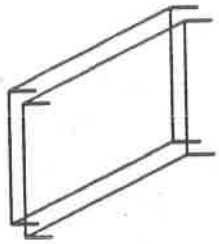

Figure 10b: Solids overlap.

3.6. IN/OUT Classification Procedure

In Step 3 of the decomposition procedure, suppose that one of the fragments to be classified is empty. Then all the faces of the other fragment can be classified directly without further decomposition. The technique for doing this is general, in that it also works for non-manifolds.

Without loss of generality, assume that $F^S_{\geq}$ and $F^T_{\geq}$ are the fragments to be classified, and that $F^S_{\geq}$ is empty. Then the interiors of the faces of fragment $F^T_{\geq}$ must be either completely inside or completely outside the solid S. If they are inside, then all the faces of $F^T_{\geq}$ should be added to the $T_{\mathrm{IN}}S$ classification set; otherwise, the faces should be added to the $T_{\mathrm{OUT}}S$ classification set. We determine which of these cases holds using the method described below. Conceptually, this method is similar to the one reported by Thibault and Naylor

[TN87], but there are some differences (for example, we handle non-manifolds, whereas Thibault and Naylor do not).

Let $p = (x, y, z)$ be any point in $R_{\geq}$, and f be any face in $F^S_<$ which is at least partially visible from p and not coplanar with p (for an example, see Figure 3.11). If (A, B, C, D) is the 4-tuple representing the plane containing f, then $R_{\geq}$ (and hence $F_{\geq}$) is inside the solid S if $Ax + By + Cz + D < 0$, and is outside otherwise. It remains only to choose a suitable point p and face f.

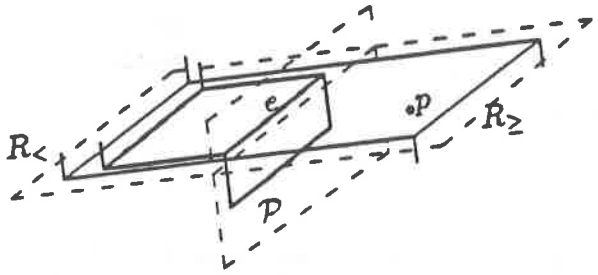

Figure 3.11: In/Out classification of planar regions

The point p can be obtained from any face of the non-empty fragment $F^T_{\geq}$. Since a point on the border of a face in $F^T_{\geq}$ can be outside $R_{\geq}$, we guarantee that p is in $R \geq$ by taking it to be the centroid of the face.

The face f is located as follows:

1. Let v be a vertex in $F^S_<$ which is closest to the plane $\mathcal{P}$. (This is the easiest way to insure that v is visible from at least one point in $R_{\geq}$.)

2. Find an edge e of $F^S_<$ incident on v such that the angle between e and the vector from v to p is minimized.

3. Find a face f incident on e such that the angle between $\overline{N}_f$ and the vector $\overline{v}$ is minimized. (This is the easiest way to insure that f is at least partially visible from at least one point in $R_{\geq}$ and is not coplanar with p.)

Note that the face f may lie outside the region R. Allowing this to occur makes the method sufficiently general that the case in which region R is planar does not have to be handled differently.

4. Implementation Considerations

Our algorithm becomes simpler if applied to the problem of intersecting collections of 2-dimensional polygons; in fact, the original development of the algorithm was for the 2-dimensional case. The algorithm was originally implemented for 2-dimensional polyhedra using the C programming language, running under the Unix operating system on Sun-2 and Sun-3 workstations [VN87a]. The algorithm was also coded into a 2-dimensional solid modeler using the SunView window environment to provide a convenient user interface.

Our algorithm has been implemented as part of a 3-dimensional solid modeler we are building called ProtoSolid. This section describes some of the implementation issues we are encountering, and how we are solving them.

4.1. Language Issues

ProtoSolid is being implemented on a TI/Explorer II, using Common Lisp [Ste84]. Lisp was the programming language of choice for several reasons:

1. The language provides the necessary attributes to set up structures as given above. Programming solid modelers is to a large extent a play on the use of pointers, lists and symbols. Lisp is a language designed to do just this.

2. The availability of specialized hardware to execute Lisp programs provides for a software environment well-suited for rapid prototyping of code. The Explorer architecture has a dedicated 36-bit processor with run-time data-type checking, a high-resolution bit-mapped display, and Ethernet based networking. The software environment includes an excellent editor with advanced features such as interpretation and compilation of code within the editor, incremental compilers, dynamic linking and loading, a flexible display-oriented debugging system and other utilities.

We now see Lisp as the obvious choice of language for solid modelers.

4.2. Data Structures

When the face-splitting procedure (Section 3.4.1) is invoked on a face f, it uses Eq. (3.4) to compute the distance $d(p)$ from p to $\mathcal{P}$ for every vertex p of f. Since

the face-splitting procedure is invoked separately for every face incident on p, this means that a naive implementation of the face-splitting procedure would compute $d(p)$ several times for each p. In our implementation, the algorithm is made more efficient by computing $d(p)$ only once and storing it in a table for later reference. Similar kinds of repetitive computations arise for other geometrical entities as well, such as points, unit vectors, and planes. Thus, we maintain a separate table for computations involving each kind of geometrical entity.

It has been commonly recognized that the separation of topological and geometrical information has computational benefits. In particular, although each solid has its own topological graph structure, all solids share the same 3-dimensional space, and thus they need to share the same geometrical entities. Thus, the tables referred to above are global to all solids.. Not only does this minimize the space for storing geometrical information, but since it eliminates redundancy among existing solids, the equality check requires only a simple address comparison, and efficient tables requiring $O(\log n)$ insertion and deletion can be used. In Protosolid, each geometrical table has been implemented as a binary search tree that uses three-way comparison functions. Tree balancing (e.g. using AVL trees) has not been necessary, as the seemingly random order of insertions rarely causes the binary search tree to degenerate [AHU74, page 118].

The splitting plane selection procedure, the fragment and face splitting procedures, and the classification procedure can be cleanly coded independently of each other. However, better efficiency can be achieved by collecting and sharing information between them. For example, the classification procedure needs to locate a fragment's closest vertex to the splitting plane. Since the splitting procedures already has looked at every vertex of the fragment, it is more efficient for the splitting procedures to retain the closest vertex for the benefit of the classification procedure.

The topological and geometrical entities of a solid are represented in a hierarchy shown in Figure 4.1. The *fedge* entity gives a face its orientation. Our use of fedges is analogous to Mäntylä's use of half-edges in the half-edge data structure, except that the need for edge loops for representing holes has been eliminated as was done in the bridge-edge data structure used by Yamaguchi [YT85]. All faces and edges are are maintained by the solid in a doubly-linked list. The vertices of a given solid are maintained in a binary search tree spatially ordered by their coordinates.

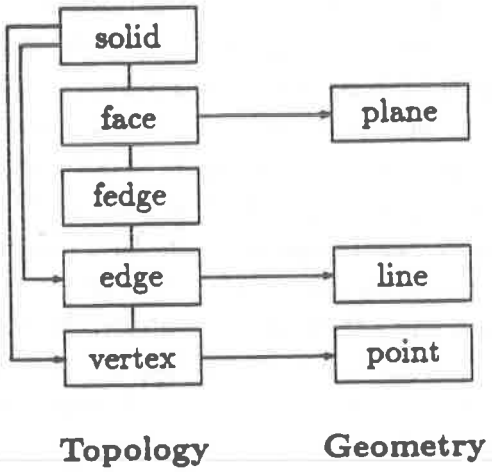

Figure 4.1: Solid Model Boundary Hierarchy.

5. Performance

The cost of the algorithm can be expressed as a recurrence relation. However, even for the 2-dimensional case, the recurrence relation is a complicated one [VN87a] which might not be feasible to solve analytically. Furthermore, it provides only a very loose bound on the cost of the algorithm. Although the worst case should be easier to determine than the average case, it has been observed that in geometric modeling systems, the worst case is generally very pessimistic and sheds little light on the average-case running time.

An alternate approach is to observe the average case running time statistically, by measuring the number of occurrences of several key operations performed by the algorithm. This was done earlier for an implementation of our algorithm on 2-dimensional polygons [VN87b]. Using an algorithm for generating random polygons of arbitrary size, several batches of 50 pairs of unique polygons were intersected and their averages plotted. Eight such batches with increasing number of edges from 4 to 512 each were performed. The algorithm's complexity was measured in this way under numerous test runs, and the average-case complexity was observed to be approximately $O(n \log n)$. We have not yet been able to do a similar test for the 3-dimensional case, because of the diffi-

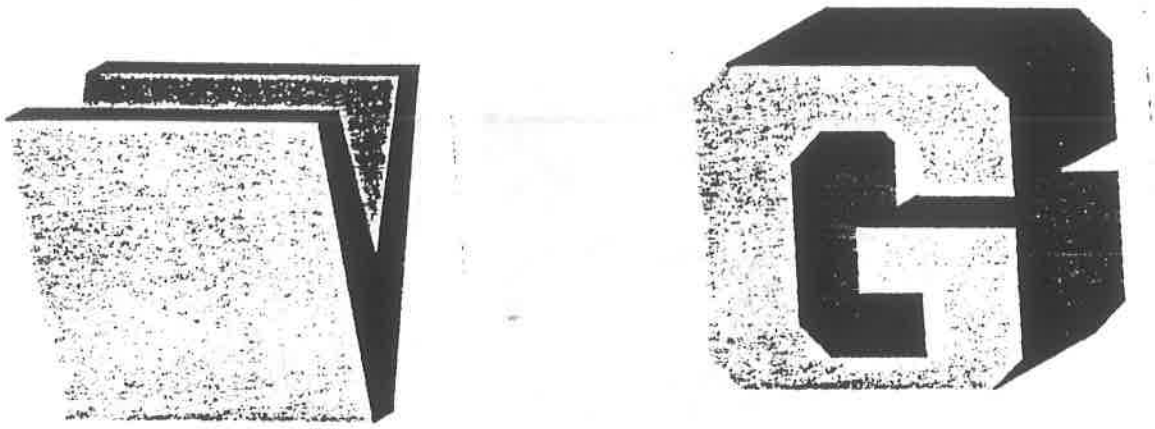

Figure 5.1: Two solids shaped like the letters V and G.

culty of generating random 3-dimensional polyhedra. However, we expect that the average-case complexity for the 3-dimensional case will be the same as for the 2-dimensional case.

As an example, we now discuss the use of the algorithm to compute the intersection of two solids. The CPU times below are taken from ProtoSolid executing on a TI/Explorer-II with non-optimized, but compiled Common Lisp code. Figure 5.1 shows two solids G and V, shaped like the letters G and V. These solids took 0.32 and 0.12 CPU seconds to create, respectively. Figure 5.2 shows $G \cap^* V$; computing this intersection took 3.62 CPU seconds. This 3.62 seconds consisted of 2.7 seconds for boundary decomposition and classification, 0.87 seconds for creating the solid representing the intersection, and 0.05 seconds for simplifying it to obtain maximal faces. During boundary classification, the top level classification routine was called 171 times. The boundary decomposition and classification step decomposed the nine original faces of V into 85 homogeneous faces, and the 23 original faces of G into 131 homogeneous faces.

In the above example, most of the time required for computing $G \cap^* V$ was for boundary decomposition and classification. When boundary decomposition and classification is done, Protosolid retains the results of this step, making it easy to compute other set operations on the same objects. For example, Figure 5.3 shows $V -^* G$. Computing this solid took 0.69 additional CPU seconds, consisting of 0.67 seconds for creating the solid and .02 seconds for simplifying it to obtain

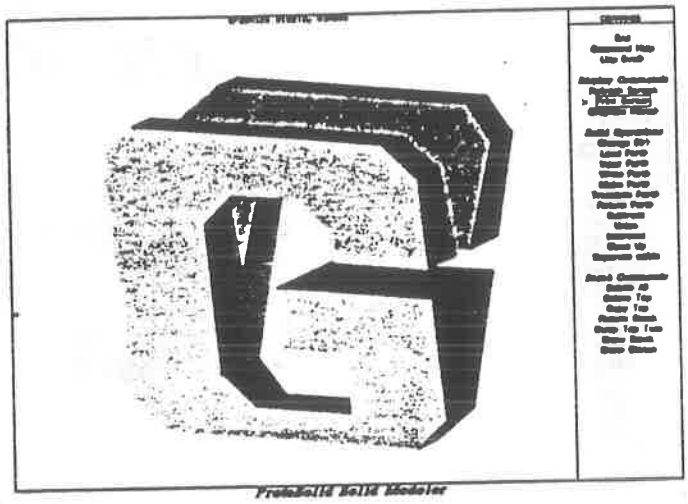

Figure 5.2: The intersection of G and V.

maximal faces.

If two solids are identical except for a small angle of rotation, many solid modelers fail to do set operations on these solids correctly. ProtoSolid is successful for rotations as small as 0.1 degrees, and most of the time succeeds for even smaller rotations.

6. Conclusion and Future Work

This paper presents a new algorithm for set operations on 3-dimensional polyhedra represented using boundary representations. This algorithm has several appealing characteristics:

1. The polyhedra need not be bounded by 2-manifolds.
2. The same algorithm can also be used for 2-dimensional polygons. (In this case, the polygons need not be bounded by 1-manifolds.)
3. The algorithm operates directly on the boundary representations, without requiring conversion to and from other representations.
4. The algorithm does not have to deal with numerous special cases as do many existing algorithms.

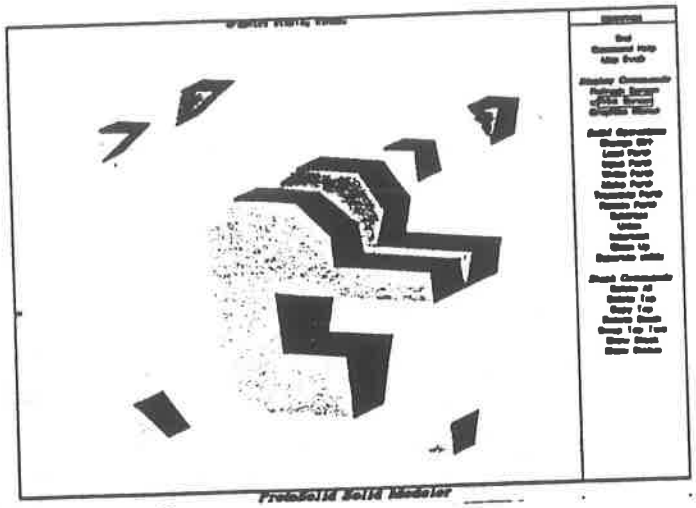

Figure 5.3: The removal of G from V.

5. The divide-and-conquer approach used by our algorithm provides better average-case performance than the quadratic cost of most existing algorithms. The running time has been empirically determined to be between $O(n)$ and $O(n^2)$.

We are currently implementing a solid modeler using this algorithm. The solid modeler, Protosolid, is implemented in Common Lisp on a TI/Explorer II. This will make it easy to integrate the modeler to systems we are developing to do geometric reasoning for automated manufacturing [NIK*88]. In connection with the ongoing implementation of Protosolid, several issues for future research arise:

1. As mentioned in Section 3.3, the splitting plane is normally chosen to be either coplanar or perpendicular to some face of the one of the solids. By analogy with the Quicksort algorithm [Hoa62], our algorithm is most efficient when the plane decomposes fragments into subfragments of equal sizes. We intend to investigate various selection strategies in detail, to determine their relative merits.

2. The time complexity of the algorithm has not yet been satisfactorily analyzed. Although the cost of the algorithm can be expressed as a recurrence relation, the recurrence relation is quite complex, and it may not be feasible to determine the complexity analytically. An alternate approach is

to observe the average case running time statistically. Although we have done this for the 2-dimensional case, it still remains to do this for the 3-dimensional case.

3. The use of Protosolid in our automated manufacturing project will require that Protosolid be extended to handle nonflat surfaces.

4. The efficiency of the implementation depends heavily on using an internal representation of the boundaries of the solids that is efficient for non-manifold boundaries. In our implementation of the algorithm, it is not clear whether we have found the most efficient data structures for this problem. Thus, we are investigating several data structures for representing the boundaries, to see which will provide the best performance.

References

[ABN85] D. Ayala, P. Brunet, and I. Navazo. Object representation by means of nonminimal division quadtrees and octrees. *ACM Transactions on Graphics*, 4(1):41–59, January 1985.

[AHU74] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, Massachusetts, 1974.

[Bau72] B. Baumgardt. *Winged-edge polyhedron representation*. Technical Report CS-320, Stanford University, Stanford, CA, 1972.

[BHS80] I. C. Braid, R. C. Hillyard, and I. A. Stroud. *Stepwise construction of polyhedra in geometric modeling*, pages 123–141. Academic Press, New York/London, 1980.

[Bro76] K. D. Brotz. Intersecting polyhedra with successive planes. *Computers & Graphics*, 2(1-A):1–6, 1976.

[FKN80] H. Fuchs, Z. M. Kedem, and B. F. Naylor. On visible surface generation by a priori tree structures. *Conf. Proc. of SIGGRAPH '80*, 14(3):124–133, July 1980.

[Hoa62] C. A. R. Hoare. Quicksort. *The Computer Journal*, 5:10–15, 1962.

[Man84a] M. Mäntylä. A note on the modeling space of euler operators. *Computer Vision, Graphics, and Image Processing*, 26:45–60, 1984.

[Man84b] M. Mäntylä. *Part Three: Advanced Toplics*. Helsinki University of Technology, 1984.

[Man86] M. Mäntylä. Boolean operations on 2-manifolds through vertex neighborhood classification. *ACM Transactions on Graphics*, 5(1):1–29, January 1986.

[Nav86] I. Navazo. *Geometric Modelling of Octree encoded Polyhedrical Objects*. PhD thesis, Universitat Politecnica de Catalunya, Departament de Metodes Informatics, January 1986.

[NIK*88] D. S. Nau, N. Ide, R. Karinthi, G. Vaněček Jr., and Q. Yang. Integrating ai and solid modeling for process planning. In *Third Internat. Conf. CAD/CAM, Robotics, and Factories of the Future*, Southfield, MI, August 1988.

[PS86] L. K. Putnam and P. A. Subrahmanyam. Boolean operations on n-dimensional objects. *IEEE Computer Graphics & Applications*, 43–51, June 1986.

[Req77] A. A. G. Requicha. *Mathematical models of rigid solid objects.* Technical Report Tech. Memo 28, Production Automation Project, University of Rochester, Rochester, N.Y., November 1977.

[Req80] A. A. G. Requicha. Representations for rigid solids: theory, methods, and systems. *ACM Computing Surveys*, 12(4), December 1980.

[RV77] Aristides A. G. Requicha and H. B. Voelcker. *Constructive solid geometry.* Technical Report 25, University of Rochester, Rochester, NY, November 1977.

[RV83] A. A. G. Requicha and H. B. Voelcker. Solid modeling: current status and research directions. *IEEE Computer Graphics & Applications*, October 1983.

[RV85] A. A. G. Requicha and H. B. Voelcker. Boolean operations in solid modeling boundary evaluation and merging algorithms. *PIEEE*, 73(1):30–44, January 1985.

[SH74] I. E. Sutherland and G. W. Hodgman. Reentrant polygon clipping. *Communications of the ACM*, 17(1):32–42, January 1974.

[Ste84] G. L. Steele Jr. *Common LISP, The Language.* Digital Press, Burlington, MA, 1984.

[Til77] R. B. Tilove. *A study of geometric set-membership classification.* Master's thesis, University of Rochester, Rochester, NY, November 1977.

[Til80] R. B. Tilove. Set membership classification: a unified approach to geometric intersection problems. *IEEE Transactions on Computers*, C-29(10):874–883, October 1980.

[TN87] W. C. Thibault and B. F. Naylor. Set operations on polyhedra using binary space partitioning trees. *ACM Computer Graphics SIGGRAPH '87*, 21(4):153–162, July 1987.

[VN87a] G. Vaněček Jr. and D. S. Nau. *Computing Geometric Boolean Operations by Input Directed Decomposition.* Technical Report TR–87–8, Systems Research Center, University of Maryland, College Park, MD, 20742, January 1987. See Also TR 1762.

[VN87b] G. Vaněček Jr.. and D. S. Nau. Non-regular decomposition: an efficient approach for solving the polygon intersection problem. In C. R. Liu, A. Requicha, and S. Chandrasekar, editors, *Intelligent and Integrated Manufacturing Analysis and Synthesis*, pages 271–280, December 1987.

[YT85] F. Yamaguchi and T. Tokieda. Bridge edge and triangulation approach in solid modeling. In Tosiyasu L. Kunii, editor, *Frontiers in Computer Graphics '84*, pages 44–65, Springer-Verlag, 1985.