6 Fleshing Out Wire Frames: Reconstruction of Objects, Part I*

George Markowsky
Dept. of Computer Science, University of Maine, Orono, Maine

Michael A. Wesley
Manufacturing Research, IBM Thomas J. Watson Research Centre, Yorktown Heights, N.Y.

ABSTRACT

We present algorithms for reconstructing solid polyhedral objects. In this chapter we present an algorithm which discovers all objects with a given wire frame. This algorithm, which has a number of applications to mechanical design besides being of mathematical interest, has been implemented and has performed well on complex objects.

1. INTRODUCTION

The application of computers to problems in mechanical design was first recognized over 20 years ago (Sutherland, 1963). Since that time much work has been done on the development of production systems both for the entry of a design into a computer data base and for the use of a mechanical design data base in design analysis and in manufacturing.

In the field of data base entry, computer drafting systems allow a designer to interact with a display or tablet to produce drawings of objects, generally in the classic manner of two-dimensional projections of the edges of the three-dimensional object. Some systems also provide the capability of representing data in three dimensions; for example, depth coordinates may be added to the elements of a two-dimensional view, corresponding features in each of several views may

Reprinted from *IBM Journal of Research and Development*, Vol. 24, No. 5, September, 1980.

be related, and isometric views may be constructed. These computer drafting systems have been engineered to very high levels of performance and can greatly enhance the productivity of a designer. As well as producing drawings of the edges of objects, computer drafting systems exist which allow the description of the surfaces of objects as smooth curves or patches splined together at their boundaries; in general these surfaces are represented by means of a discrete mesh superimposed on the surface.

Computer-based systems are also used in the analysis of designs and in the manufacture and assembly of objects. For example, parts can be checked for interference (Boyse, 1979; Wesley et al., 1980), finite element methods may be used for analysis of, for example, heat flow (Brown, 1977), the constraints between objects can be derived and mechanisms can be simulated (Taylor, 1976), numerically controlled machine tool tapes can be generated to allow manufacture of a part (Woo, 1975), and robot motions to assemble parts can be generated (Lozano-Perez & Wesley, 1979; Udupa, 1977). In general the full automation of these applications of the design data base requires three-dimensional volumetric information about an object, rather than just a description in terms of edges and surface mesh facets. At present the volumetric form of the data base is considered to be rather difficult and expensive to acquire, and the analysis data are generated in a computer-assisted manner. For example, in the case of numerically controlled machine tools, the path of the cutter may be entered over a drawing at a graphics terminal.

This paper presents an algorithm for automatically bridging the gap between these two fields of computer geometry, that is, from an object described in terms of its edges (a *wire frame*) to a volumetric description in terms of solid material, empty space, and the topology of surfaces and edges. In its present form, the algorithm is restricted to objects whose edges are straight lines and whose faces are planar; since the algorithm is a topological algorithm, it could be adapted to nonplanar surfaces.

Quite apart from its practical applications, the problem is of some theoretical interest. An edge description does not necessarily represent a unique object, and an algorithm should be able to detect ambiguities, enumerate solutions, and accept user decisions as to which solution is required. As with many geometrical problems, the simple cases are straightforward and the complex cases are extremely difficult; for example, many pathological cases can exist—vertices and edges contacting faces, and coplanar opposing faces meeting with edge contact.

Although the literature on geometric modeling is extensive (Baer, Eastman, & Henrion, 1979) and growing rapidly, few authors have chosen to represent objects formally. They are therefore generally unable to prove the correctness of their methods, to handle the full range of pathological cases and ambiguities that occur in practice, or even to describe objects precisely. However, the PADL project (Requicha & Tilove, 1978) is based on point set topology and its architects are able to prove the correctness of algorithms for computing the set operations of union, intersection, and difference between polyhedra. Other workers

have used Euler operators (Baer et al., 1979) to ensure correctness of topology as an object is constructed. Idesawa (Idesawa, 1973; Idesawa, Soma, Goto, & Shibata, 1975) describes a wire frame reconstruction scheme as part of the general problem of constructing solids from many 2-D projections. The method used is based on finding sets of coplanar edges and fitting them together to form solid objects; however, the reconstruction method is not based on a formal description of objects and does not handle ambiguities or many of the pathological cases. Experience in modeling has taught us that even though pathological relationships among faces, edges, and vertices may not be physically realizable, they do occur frequently in the stylized world of geometric modeling, and a general-purpose modeling system should be able to handle them. Lafue (1976) also describes a program for generating solids from 2-D projections, but requires that objects be described in terms of faces rather than edges; further, faces are described in a stylized manner with extra edges to permit description of holes.

Other authors have considered the machine vision problem of recognizing polyhedral objects from incomplete edge descriptions (Clowes, 1971; Huffman, 1971; & Waltz, 1975). In this situation local ambiguities can exist and are resolved, if possible, by global propagation. The propagation is performed by labeling areas, whereas the algorithm described in this paper handles ambiguities in terms of volume regions. The use of volumes rather than areas leads to a much simpler handling of the process of labeling.

This paper is divided into four sections. Section 2 gives formal definitions of the concepts needed in order to be able to explain the algorithm and describes some of their consequences. Some standard topological notation is discussed in the appendix. Hocking and Young (1961) may be used as a reference for these terms. Section 3 describes the stages of the algorithm, which has been coded and has performed well even on complex objects. Section 4 gives a number of examples which illustrate the performance of the algorithm.

2. BASIC CONCEPTS

The concepts defined in this section are based on some fundamental topological ideas which are described in detail in Hocking and Young (1961) and to a lesser extent in the appendix. Throughout the paper the standard topology in $\mathbb{R}^3$ and the induced topology on subsets of $\mathbb{R}^3$ are assumed. Vertices refer to points in $\mathbb{R}^3$ and edges refer to line segments defined by two points in $\mathbb{R}^3$. The approach used in this section is to define faces, objects, and wire frames, and then describe the consequences of these definitions.

Definition 1

A *face*, f, is the closure of a nonempty, bounded, connected, coplanar, open (in the relative topology) subset of $\mathbb{R}^3$ whose boundary (denoted by ∂f) is the union of a finite number of line segments. P_f is used to denote the unique plane which contains f.■

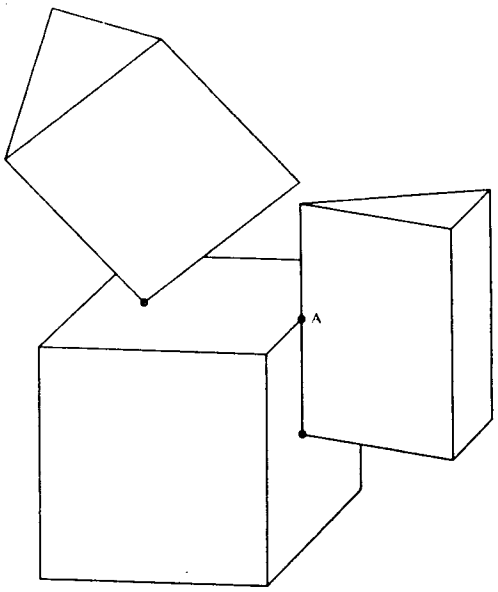


FIG. 6.1. An object exhibiting various kinds of intersections.

Definition 2

An *object*, $\mathbb{O}$, is the closure of a nonempty, bounded, open subset of $\mathbb{R}^3$ whose boundary (denoted by $\partial\mathbb{O}$) is the union of a finite number of faces.■

The wire frame algorithm uses many geometric facts about objects. However, rather than define an object as being a set of points satisfying a long list of properties, we have preferred to offer a very simple definition of an object and then prove that it has all the desired properties. Thus from the definitions above it is easy to see that the "cube," $\{x, y, z \in \mathbb{R}^3 | 0 \le x \le 1, 0 \le y \le 1, 0 \le z \le 1\}$ is an object and that $\{(1, y, z) \in \mathbb{R}^3 | 0 \le y \le 1, 0 \le z \le 1\}$ is one of its "square" faces. Starting off with open sets means that faces and objects have nontrivial interiors.

Notice that it is not assumed that an object is the closure of a connected set. This allows objects that consist of disjoint "solids" or even objects which intersect in edges, etc. One can argue that this last case, illustrated in Fig. 6.1. does not represent a "real" object, but in practice all sorts of strange objects can appear. Thus, we decided to handle the most general case possible. Furthermore, this generality does not exact any penalty other than creating a larger number of solutions.

Another point worth noticing is that Definitions 1 and 2 allow many different representations of the boundaries of faces and objects by line segments and faces (respectively). However, canonical representations of the boundaries can be defined which correspond to one's intuitive notions about such things. To get to these representations it is necessary to introduce several additional concepts.

Definition 3

(a) Let f be a face. The *vertices* of f, V(f), are defined to be the set of all points for which two noncollinear line segments, contained in ∂f, can be found whose intersection is the given point.

(b) Let f be a face. The *edges* of f, E(f), are defined to be the set of all line segments e, contained in ∂f, satisfying the following conditions:

1. The endpoints of e belong to V(f);
2. No interior point of e belongs to V(f).

(c) Let $\mathbb{O}$ be an object. The *vertices* of $\mathbb{O}$, V($\mathbb{O}$), are defined to be the set of all points p for which faces $f_1, f_2, f_3 \subseteq \partial\mathbb{O}$ can be found such that $f_1 \cap f_2 \cap f_3$ and $P_{f_1} \cap P_{f_2} \cap P_{f_3}$ are both exactly the single point p.

(d) Let $\mathbb{O}$ be an object. The *edges* of $\mathbb{O}$, E($\mathbb{O}$), are defined to be the set of all line segments e, contained in $\partial\mathbb{O}$, satisfying the following conditions:

1. The endpoints of e belong to V($\mathbb{O}$);
2. No interior point of e belongs to V($\mathbb{O}$);
3. For every point p of e, two noncoplanar faces can be found, $f_1, f_2 \subseteq \partial\mathbb{O}$ such that $p \in f_1 \cap f_2$.

(e) Let $\mathbb{O}$ be an object. The *wire frame* of $\mathbb{O}$, WF($\mathbb{O}$), is defined to be the ordered pair [V($\mathbb{O}$), E($\mathbb{O}$)]. □

The concepts have been defined, but some work is required to show that things fit together as expected. For example, it is not clear that V($\mathbb{O}$) is finite. The reason for keeping the definitions so general is that it is fairly easy to check whether the concepts here include a particular class of entities.

It can be shown that V(f), E(f), V($\mathbb{O}$), and E($\mathbb{O}$) are all finite. V(f) and E(f) yield the intuitive representation of f which will be described more fully below. V($\mathbb{O}$) and E($\mathbb{O}$) do not quite represent $\mathbb{O}$, since the faces of $\mathbb{O}$, which have not been defined so far, are needed. Before getting into the definition of the faces of $\mathbb{O}$, the nature of faces is first described in somewhat greater detail. To do this an additional concept is needed.

Definition 4

A *1-cycle* is a collection of coplanar line segments $\{e_1, \ldots, e_k\}$ in $\mathbb{R}^3$ having the following properties:

1. The intersection of two distinct elements e_i and e_j is either the empty set or a point which is an endpoint of both line segments;
2. Every point of $\mathbb{R}^3$ is the endpoint of a *nonnegative, even* number (in most cases 0) of the e_i. □

In order to be able to give a complete definition of a face, it is necessary only to describe what is meant by the inside and outside of a 1-cycle. A 1-cycle has an inside (outside) if there is a bounded (unbounded), connected, open set whose boundary is the given 1-cycle. A 1-cycle may lack an inside or an outside, or may have them both. Some examples are discussed following Theorem 5. Whenever a point is said to be inside (outside) a 1-cycle, $\mathcal{C}$, it is meant that $\mathcal{C}$ has an inside (outside) and that the point in question belongs to some bounded (unbounded), connected, open set whose boundary is $\mathcal{C}$. Actually, if a 1-cycle, $\mathcal{C}$, has an inside (outside), there is a component of the complement of $\mathcal{C}$ whose boundary is $\mathcal{C}$ and which contains all other bounded (unbounded), connected, open sets whose boundary is $\mathcal{C}$.

Theorem 5

Let f be a face. Then 1-cycles, $\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_k$ $(k \geq 0)$, contained in ∂f can be found such that

1. $\partial f = \mathcal{C}_0 \cup \mathcal{C}_1 \cup \ldots \cup \mathcal{C}_k$;
2. Face f consists of all points inside $\mathcal{C}_0$ and outside

$$\mathcal{C}_i \ (i \geq 1), \text{ and the points of } \bigcup_{i=0}^{k} \mathcal{C}_i;$$

3. The 1-cycles are all disjoint. □

A typical face is pictured in Fig. 6.2. Note that the boundary can intersect itself at various points such as v_1, v_{16}, and v_{18}. The face in Fig. 6.2 can be defined in terms of three 1-cycles: $\mathcal{C}_0$ (traced out by following the sequence of vertices $v_1\ v_2 \ldots v_9\ v_{10}\ v_1\ v_{11}\ v_{12}\ v_{13}\ v_1$), $\mathcal{C}_1$ (traced out by $v_{14}\ v_{15}\ v_{16}\ v_{17}\ v_{18}\ v_{19}\ v_{20}\ v_{18}\ v_{16}\ v_{14}$), and $\mathcal{C}_2$ (traced out by $v_{21}\ v_{22} \ldots v_{28}\ v_{21}$). Thus the face in Fig. 6.2 consists of all points in the inside of $\mathcal{C}_0$ and in the outsides of $\mathcal{C}_1$ and $\mathcal{C}_2$, plus the points actually belonging to $\mathcal{C}_0$, $\mathcal{C}_1$, and $\mathcal{C}_2$. Note that the points v_2, v_1, and v_{11} are collinear, but by the definition of E(𝕆), $v_2\ v_1$ and $v_1\ v_{11}$ actually belong to E(𝕆), but $v_2\ v_{11}$ does not. Note also that not every 1-cycle has an inside ($\mathcal{C}_1$, for example, fails the connected set requirement). Similarly, not every 1-cycle has an outside ($\mathcal{C}_0$, for example, again fails the connected set requirement).

The faces of an object are now defined. Again, the definition turns out to be rather straightforward, and it can be shown that the faces defined in this way actually turn out to be the things one would like to call faces anyway. From the following definition it is clear that the faces of an object are really determined by the object rather than by any particular representation of the object or its boundary.

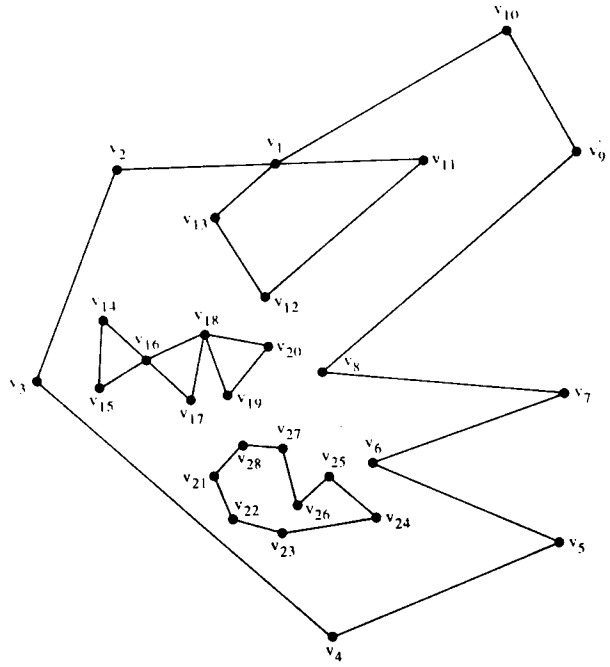


FIG. 6.2. A typical face.

Definition 6

Let 𝕆 be an object. The *faces* of 𝕆, F(𝕆), are defined to be the closures of the connected components of $\partial\mathbb{O} - \cup$ E(𝕆). □

A number of the results hold true for the faces of an object. Some of the important relationships are summarized in the following theorem.

Theorem 7

Let 𝕆 be an object and F(𝕆) = $\{f_1, \ldots, f_m\}$. Then

1. $\partial\mathbb{O} = \bigcup_{i=1}^{m} f_i$;

2. $\bigcup_{i=1}^{m} E(f_i) \subseteq E(\mathbb{O}) \cup \Gamma$, where Γ is the set consisting of all line segments which are unions of elements of E(𝕆);

3. $\bigcup_{i=1}^{m} V(f_i) \subseteq V(\mathbb{O})$;

4. Any face $f \subseteq \partial\mathbb{O}$ for which $E(f) \subseteq E(\mathbb{O}) \cup \Gamma$, where Γ is as in (2) above, is the union of elements of $F(\mathbb{O})$;

5. The intersection of two distinct elements of $F(\mathbb{O})$ is a union of elements of $E(\mathbb{O})$ and subsets of $V(\mathbb{O})$. □

A few brief remarks about (2) and (3) above are in order. Vertices of an object need not be vertices of any face; e.g., in Fig. 6.1 point A is in $V(\mathbb{O})$ but is not a vertex of any face of $\mathbb{O}$. Thus the corresponding edges must be broken up into smaller line segments when considered as edges of $\mathbb{O}$, but this division does not occur if any face is considered separately.

To give the reader an idea of what objects look like in this model, two additional concepts are needed.

Definition 8

A *primitive object* is an object whose interior is connected. □

The key point of Definition 8 is to prevent problems caused by the peculiar types of intersections illustrated in Fig. 6.1. In that figure, there are three primitive objects: a cube and two triangular prisms. In general, an object can be decomposed into primitive objects which do not have any 2-dimensional intersections between any two of them.

Definition 9

A 2-cycle is a collection of faces $\{f_1, \ldots, f_m\}$ in $\mathbb{R}^3$ having the following properties:

1. The intersection of two distinct elements f_i and f_j is the union of the elements of $E(f_i) \cap E(f_j)$ and a finite number of points;

2. Every element of $\bigcup_{i=1}^{m} E(f_i)$ belongs to an even number of distinct faces. □

With all of these concepts a description of primitive objects can be given in terms of 2-cycles. From this one can extrapolate to objects in general. Note how similar the description is to the one given for faces. The definitions of inside and outside are similar to those defined for 1-cycles.

Theorem 10

Let $\mathbb{O}$ be a primitive object. Then 2-cycles $\mathscr{C}_0, \mathscr{C}_1, \ldots, \mathscr{C}_k$ $(k \geq 0)$ contained in $\partial\mathbb{O}$ can be found such that

1. $\partial\mathbb{O} = \mathscr{C}_0 \cup \mathscr{C}_1 \cup \ldots \cup \mathscr{C}_k$;

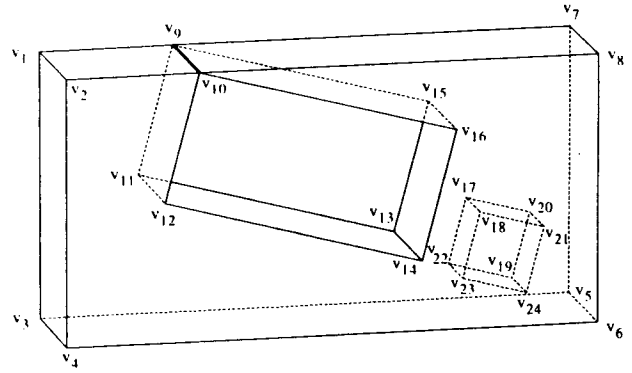


FIG. 6.3. A typical primitive object.

2. $\mathbb{O}$ consists of all points inside $\mathscr{C}_0$ and outside $\mathscr{C}_i$ $(i \geq 1)$ and the points in $\bigcup_{i=0}^{k} \mathscr{C}_i$;

3. The 2-cycles are all disjoint. □

The term cycle is used below in situations where it is clear from the context whether a 1-cycle or a 2-cycle is intended. Furthermore, the comment made to the effect that not all 1-cycles have a well-defined inside or outside applies as well to 2-cycles, where inside and outside are defined in a similar manner. It will be seen in the next section that to recover $\mathbb{O}$ from $WF(\mathbb{O})$ will be necessary to decompose primitive objects further.

Figure 6.3 illustrates a typical object which is represented by two cycles $\mathscr{C}_0$ and $\mathscr{C}_1$. The exterior cycle, $\mathscr{C}_0$, consists of 11 faces, while the interior cycle, $\mathscr{C}_1$, has 6. The identification of the faces is left to the reader.

3. THE WIRE FRAME ALGORITHM

The goal of the wire frame algorithm is to construct all objects which have a given wire frame. It is a fairly elaborate algorithm with quite a few distinct stages. The key stages of the algorithm are outlined first below, followed by a more detailed description.

Stages of the Algorithm

1. Checking Input Data. The input data are assumed to be a valid wire frame, that is, the ordered pair of vertices and edges [V(𝕆), E(𝕆)] (Definition

3(e), above). In this stage the input data may be checked for various kinds of errors, such as nondistinct vertices and edges. The choice of actual tests performed is based on the source of the input data and the expected types of errors.

2. *Finding Planar Graphs.* All planes are found which contain at least two intersecting edges. For each distinct plane a canonical normal is defined and a graph of coplanar edges formed. For each vertex lying in a plane, a circular list of edges meeting that vertex is created and ordered counterclockwise with respect to the canonical normal.

3. *Calculation of 1-Cycles and Virtual Faces.* In each planar graph the set of partitioning cycles is uncovered (bridges are ignored). The nesting relationships among these cycles are then determined, and all candidates for faces found. These candidates are called *virtual faces.*

4. *Checking for Illegal Intersections Between Virtual Faces.* Two virtual faces can intersect illegally, i.e., so that both cannot be faces of the real object, in only two ways. These intersections are detected in this stage and appropriate action taken:

> A type I intersection occurs when an interior point of an edge of one pierces an interior point of the other. The latter virtual face is deleted.
>
> A type II intersection occurs when there is no type I intersection, yet a vertex of one is in the plane of the other and there exists a point that is interior to both. A decision on the faces cannot be made at this stage, and temporary additional edges called *cutting edges* are introduced. These cutting edges cut some of the virtual faces discovered in Stage 3 into new, smaller, virtual faces.

5. *Calculation of 2-Cycles and Virtual Blocks.* For each edge a circular list of the virtual faces containing that edge is created. This list is ordered radially around the edge. These lists are used to find all partitioning cycles of the virtual face graph; the nesting relationships among these cycles are found and used to uncover all candidates for solid regions. These candidates are called *virtual blocks.* Virtual blocks are bounded by virtual faces and partition IR^3. Any virtual face which does not belong to two different virtual blocks is dropped.

6. *Constructing All Solutions for the Wire Frame.* A decision tree, based on virtual blocks and using a few basic tests, assigns solid or hole state to all virtual blocks and thereby constructs all possible objects having a given wire frame. In this decision process, edges and cutting edges are treated separately; cutting edges are subsequently removed. □

The reader should keep in mind that the above description and the one below are designed for easy comprehension. As a result descriptions of each of the

stages are given without describing every detail of the data structures and algorithms used. A more detailed description of the various stages follows.

Stage 1: Checking Input Data

The input to the wire frame algorithm must be a valid wire frame, that is, the ordered pair of vertices and edges [V(𝕆), E(𝕆)] (Definition 3(e), above). This input is assumed to be in the form of a list of vertices with their 3-dimensional coordinates and a list of pairs of vertices to represent edges. The wire frame algorithm described in the following sections requires that the input data represent a valid wire frame, that is, a wire frame that satisfies the definitions of edges and vertices given in Section 2. In this stage tests are performed to check the validity of the input data and to obtain information to be used in later stages. The exact choice of which tests to include depends on the characteristics of the input data and performance trade-offs between the cost of performing a test first, the usefulness of information generated for later stages, and the desirability of reporting errors before incurring the cost of executing the algorithm. These issues are not considered further here.

Two fairly straightforward tests check that vertices and edges are distinct and correctly defined. Furthermore, throughout the rest of the paper it is assumed that each vertex and edge has a unique index.

Another test ensures that every vertex belongs to at least three edges (this is a consequence of the definitions). This test is organized so that a table is generated showing which edges belong to which vertex. This table is important and will be used below.

A test which might also be performed at this point consists of checking that edges intersect only at endpoints, i.e., in elements of V(𝕆). Since two line segments can intersect only if they are coplanar, this test can also partition line segments into coplanar sets. Furthermore, it can even produce a list of edges which intersect a given plane. A test designed to work on the idea just put forth could be fairly expensive in terms of computer time (worst case $O[E(\mathbb{O})^2]$). Alternate tests are possible which are quicker but yield less information.

Depending on the operating environment, one can omit any of the above tests or substitute others if necessary.

Stage 2: Finding Planar Graphs

In this stage all planes which contain at least two intersecting edges are found, and for each plane a graph is constructed of the edges and vertices in that plane. For each vertex in WF(𝕆), a list is formed of the edges for which the vertex is an endpoint. For each noncollinear pair of edges in the list, the plane containing the edge pair is computed and a list formed of distinct planes at the vertex (each plane in IR^3 is specified uniquely once a normal and a distance from the origin along that normal are given). For each distinct plane at a vertex, a list is formed of edges in the plane for which the vertex is an endpoint, and the edges are sorted

around the normal in a counter-clockwise direction. It is now straightforward to match up planes at vertices and, for each globally distinct plane, to form graphs of the edges and vertices contained in the plane. In practice, the number of edges at a vertex is quite small so the above procedure works quickly.

Thus, the output of this stage is a list of plane equations and, for each plane, graphs of the edges and vertices in the plane.

Stage 3: Calculation of 1-Cycles and Virtual Faces

In this stage each planar edge and vertex graph is processed to find all subgraphs that could represent faces in accordance with Definitions 1 and 3. These subgraphs are candidates for faces of the object and are called virtual faces.

From the discussion in Section 2, it is clear that virtual faces can be located by finding 1-cycles and determining the various nesting relationships among these 1-cycles. To make the discussion clearer, assume a plane P and a graph formed from the edges and vertices of $\mathcal{O}$ which lie in P. The edges of the graph are of two types: bridges and nonbridges. An edge is a nonbridge if and only if it lies on some cycle. In principle, bridges must be removed. The remaining edges can then be divided up into 1-cycles which partition the plane into regions so that any face of $\mathcal{O}$ lying in P is one of the regions. In practice 1-cycles are found and bridges removed in the same operation.

The algorithm proceeds by uncovering the cycle structure of the edges in P. The methods used are now described. It can be shown that the complement of the edges in P, Γ, is an open set with a finite number of open connected components. The number of connected components is the same if the bridges are removed. Every edge which is not a bridge belongs to the closures, in fact boundaries, of two distinct components of Γ. Since the edges are to be used to form 1-cycles to bound the various components of Γ, some conventions are needed for connecting edges and the components they belong to. For the time being bridges are ignored.

Let $e = v_i\, v_j$ be an edge. There are two ways to traverse e: either from v_i to v_j or from v_j to v_i. Suppose $i < j$, and write $+e$ to denote e traversed from v_i to v_j and $-e$ to denote e traversed from v_j to v_i. Since P has a normal defined on it, whenever an edge is traversed in some direction, left and right sides of the edge can be defined as if one were walking in the same direction above the plane in the positive normal direction.

Let Γ_1 and Γ_2 be the two components of Γ whose boundaries (denoted by $\partial\Gamma_1$ and $\partial\Gamma_2$) contain e. Γ_1 is defined to be on the left, traversing e from v_i to v_j if $+e \in \partial\Gamma_1$. In this case $-e \in \partial\Gamma_2$. Similarly, if $+e \in \partial\Gamma_2$, then $-e \in \partial\Gamma_1$. This is the notation of algebraic topology.

At this point it is probably helpful to illustrate some of these ideas. Figure 6.4 shows a typical graph in a plane. This particular graph consists of 19 vertices and 23 edges. The only bridges are e_6, e_{22}, and e_{23}. Note that the bridges are in the

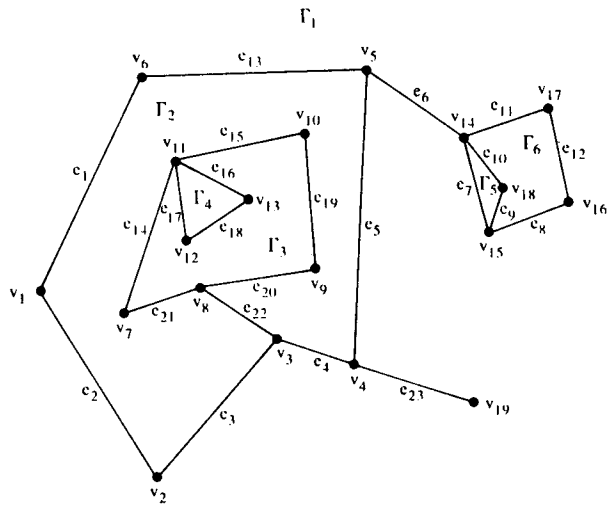


FIG. 6.4. A typical graph in a plane.

closure of exactly one component and are thus not in the boundary of any component. In this case Γ has 6 components and the following relations hold:

$$\partial\Gamma_1 = +e_1 - e_{13} - e_5 - e_4 - e_3 - e_2 + e_{11} - e_{12} - e_8 - e_7;$$
$$\partial\Gamma_2 = -e_1 + e_2 + e_3 + e_4 + e_5 + e_{13} - e_{21} + e_{14} - e_{15} - e_{19} - e_{20};$$
$$\partial\Gamma_3 = +e_{21} + e_{20} + e_{19} + e_{15} + e_{16} - e_{18} - e_{17} - e_{14};$$
$$\partial\Gamma_4 = +e_{17} + e_{18} - e_{16};$$
$$\partial\Gamma_5 = +e_7 + e_9 - e_{10};$$
$$\partial\Gamma_6 = +e_8 + e_{12} - e_{11} + e_{10} - e_9.$$

Now consider the various algorithmic steps needed to determine the information above. First pick an ordered edge, i.e., an edge and a direction, say $+e_1$; now attempt to discover the component, Γ_i, for which $+e_1 \in \partial\Gamma_i$, i.e., try to complete a cycle starting from v_1. Start at v_1 and move to v_6, pick the "next" (in a clockwise direction) edge at v_6, which is $-e_{13}$. At v_5 pick $+e_6$, then $+e_{11}$, then $-e_{12}$, then $-e_8$, then $-e_7$, then $-e_6$. Edges are checked off as they are added to a cycle; if an edge occurs twice in the same cycle, then a bridge exists (in this case e_6).

Whenever a bridge is found, there is a cycle between its two occurrences (in this case $+e_{11} - e_{12} - e_8 - e_7$). This cycle ($\mathcal{C}_1$) is set aside and the search resumed at v_5, ignoring e_6 which is removed from the graph. The sequence is now $+e_1 - e_{13} - e_5 + e_{23} - e_{23}$. The bridge detector now spots e_{23} as a bridge and removes it. The cycle between the two occurrences is the empty cycle, so the search is resumed at v_4. Another cycle ($\mathcal{C}_2$) is found as $+e_1 - e_{13} - e_5 - e_4 - e_3 - e_2$.

Each cycle is now examined to see whether the component it bounds is inside or outside of it, i.e., every point of the component is inside or outside the cycle. In this case Γ_1 is outside both $\mathcal{C}_1$ and $\mathcal{C}_2$. This information is recorded in the cycle tree described below. Also, for each vertex encountered, the cycles to which it belongs are recorded, and for each edge used, the sense in which it is used.

Now pick any other edge which has not been traversed in both directions and start all over again. Suppose $-e_{22}$ is picked now, giving the sequence $-e_{22}$ $+e_4$ $+e_5$ (e_{23} has already been eliminated) $+e_{13}$ $-e_1$ $+e_2$ $+e_3$ $+e_{22}$, at which point the bridge e_{22} is eliminated, leaving the cycle $+e_4$ $+e_5$ $+e_{13}$ $-e_1$ $+e_2$ $+e_3$, which bounds a bounded (inside) component. Data are recorded as before and the process repeated with another edge which has not yet been used in both senses. In this way the following cycles are found:

$$
\begin{aligned}
\mathcal{C}_1(\text{out}) &= +e_{11} -e_{12} -e_8 -e_7;\\
\mathcal{C}_2(\text{out}) &= +e_1 -e_{13} -e_5 -e_4 -e_3 -e_2;\\
\mathcal{C}_3(\text{in}) &= +e_4 +e_5 +e_{13} -e_1 +e_2 +e_3;\\
\mathcal{C}_4(\text{out}) &= +e_{14} -e_{15} -e_{19} -e_{20} -e_{21};\\
\mathcal{C}_5(\text{in}) &= -e_{14} +e_{21} +e_{20} +e_{19} +e_{15} +e_{16} -e_{18} -e_{17};\\
\mathcal{C}_6(\text{in}) &= -e_{16} +e_{17} +e_{18};\\
\mathcal{C}_7(\text{in}) &= +e_7 +e_9 -e_{10};\\
\mathcal{C}_8(\text{in}) &= -e_9 +e_8 +e_{12} -e_{11} +e_{10}.
\end{aligned}
$$

The notation "(in)" above shows that when the cycle is traversed in the direction indicated, the unique connected component of the complement in P of the cycle, which always lies to the left, is bounded. Similarly, "(out)" denotes the case when the cycle is unbounded.

The amount of checking that must be done may be reduced. Suppose that a bridge or generalized bridge (i.e., a connected sequence of bridges) runs between two different cycles $\mathcal{C}$ and $\mathcal{C}'$ (the sense of the bridge must agree with that of the cycles). Then at least one of them is an (out) cycle. Also, if $\mathcal{C}$ is an (out) cycle and $\mathcal{C}'$ is a distinct cycle which intersects $\mathcal{C}$ (i.e., has at least a vertex in common), then it must be an (in) cycle.

Thus in the above example, once $\mathcal{C}_1$ and $\mathcal{C}_2$ are both found to be (out) cycles, the senses of the other cycles are determined if they are derived in the sequence shown. In particular, $\mathcal{C}_3$ is an (in) cycle because it intersects $\mathcal{C}_2$. $\mathcal{C}_4$ is an (out) because it is joined by a bridge (e_{22}) to the (in) cycle $\mathcal{C}_3$. $\mathcal{C}_5$ and $\mathcal{C}_6$ are both (in) cycles because they intersect the (out) cycle $\mathcal{C}_4$, while $\mathcal{C}_7$ and $\mathcal{C}_8$ are both (in) cycles because they intersect the (out) cycle $\mathcal{C}_1$.

At this stage the cycles in P have been found, and will be used to find candidates for faces, i.e., virtual faces. The description of a face given in Section 2 and the concepts introduced here show that a face is given by its outer boundary which is an (in) cycle, $\mathcal{C}_0$, and some finite number of disjoint (out) cycles, $\mathcal{C}_1$, $\mathcal{C}_2$, ..., $\mathcal{C}_k$ which are contained in the inside of $\mathcal{C}_0$ and have the additional property that if any of them is contained in the inside of any other (in) cycle $\mathcal{C}'$, then $\mathcal{C}_0$ is contained in the inside of $\mathcal{C}'$ as well. This leads to consideration of the following tree structure.

The root is labeled by P. A cycle is a descendant of an (in) cycle, $\mathcal{C}$, if and only if it is contained in the inside of $\mathcal{C}$. A cycle is a descendant of an (out) cycle, $\mathcal{C}$, if and only if it is contained in the complement of the outside of $\mathcal{C}$. The tree structure for the cycles derived from Fig. 6.4 is given in Fig. 6.5.

A few observations aid in the construction of the tree. Any cycle which intersects an (out) cycle is automatically an (in) cycle and a son of the given cycle in the tree. Furthermore, at the finish (in) and (out) cycles must alternate. From the tree it is easy to determine that there are exactly five virtual faces at this stage: the regions bounded by $\mathcal{C}_5$, $\mathcal{C}_6$, $\mathcal{C}_7$, and $\mathcal{C}_8$; the region inside of $\mathcal{C}_3$ and outside of $\mathcal{C}_4$.

There is another point which is appropriate to bring up here. The wire frame algorithm has the property that if it fails to find an object having a given wire frame, then no such object exists. In practice, one works with wire frames of objects that exist. Thus if the final results of the algorithm indicate that no such objects exist, it is probable that some error was made in the input wire frame. Thus at the various stages there are a number of simple checks which can be performed to determine whether or not the wire frame is valid. At the end of Stage 3 one can check to see whether each edge belongs to at least two non-coplanar faces and that each vertex belongs to at least three faces which lie in planes whose intersection is exactly the vertex. Failure to meet any of these conditions would indicate the existence of an error at this point.

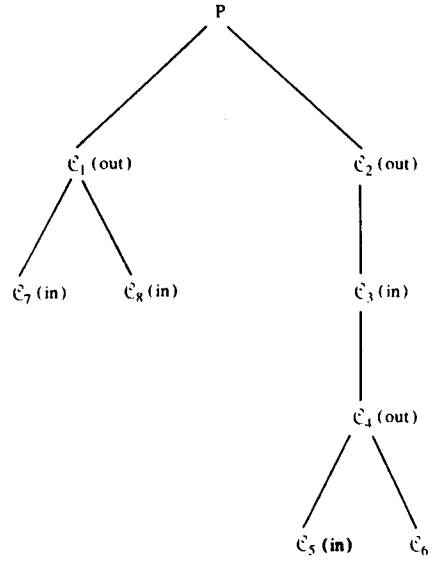


FIG. 6.5. Tree of 1-cycles.

Stage 4: Checking for Illegal Intersections Between Virtual Faces

The description of objects in Section 2 is based on 2-cycles, which have the property that the faces belonging to them intersect only at boundary points of the faces. Two virtual faces intersect illegally when there exists a point in the intersection that is internal to both. In this case it is not possible for both to be real faces of the object. Illegal intersections can occur in either of two ways:

I. An interior point of an edge of one contains an interior point of the other;
II. The above type of intersection does not occur, yet a vertex of one is in the plane of the other, and there exists a point that is interior to both faces.

These illegal intersections, which are known as type I and type II intersections, respectively, are detected in this stage, and appropriate action taken.

A type I intersection occurs when any inside point of any virtual face is an inside point of any element of $E(\mathbb{O})$. If such a condition is found, the virtual face is dropped from the list of virtual faces because it is impossible for it ever to be a face. To see this, note that the edges of $\mathbb{O}$ belong to actual faces. If a virtual face intersects edges as described above, it would have to intersect the corresponding faces. Such an intersection would produce at least one edge emanating from an *inside* point of a face, which would be impossible. (A type I intersection is shown later in Table 6.1(d).)

Two ways are proposed to handle the second case. The second method, which is the preferred method, also suggests a quick means for checking for type II intersections.

The first method of handling type II intersections is to pick maximal subsets of virtual faces which lack a type II intersection and to proceed through the remaining stages of the wire frame algorithm to uncover all possible solutions under those assumptions. In some cases one can use any solutions found to resolve the true nature of type II intersections. In other cases it might be necessary to go through the remaining stages of the algorithm with several different maximal subsets lacking a type II intersection. For many practical objects, type II intersections are relatively rare (they arise from high degrees of symmetry), so this solution is quite a practical one. It also has the advantage that it simplifies the decision procedure in Stage 6, since there is only one kind of edge to consider.

The second method is based on the observation that a type II intersection consists of a finite number of line segments, the endpoints of which are elements of $V(\mathbb{O})$. To see this, let f_1 and f_2 be faces that have type II intersection. Let $l = P_{f_1} \cap P_{f_2}$. Let $p \in f_1 \cap f_2$ be an interior point of both f_1 and f_2. Let p_1 and p_2 be the points of l which give the maximal line segment containing p and contained in $f_1 \cap f_2$. Since f_1 and f_2 are compact, i.e., closed and bounded, p_1 and p_2 belong to $\partial f_1 \cup \partial f_2$. Since no boundary point of f_1 is an inside point of f_2 and

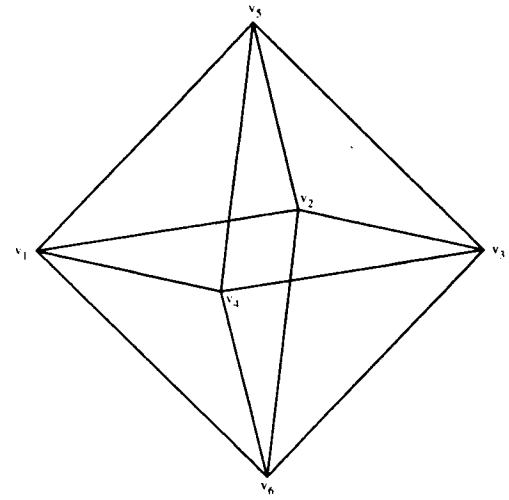


FIG. 6.6. A regular octahedron exhibits many type II intersections.

vice versa, $p_1, p_2 \in \partial f_1 \cap \partial f_2$. If the edges of f_1 and f_2 which contain p_1 (p_2) are collinear, then f_1 and f_2 must be coplanar and must overlap in nontrivial ways. This is impossible in view of the tests performed in Stage 3. Thus p_1 and p_2 belong to two noncollinear edges which can only intersect in an element of $V(\mathbb{O})$. To help visualize the preceding argument, look at Fig. 6.6. Here f_1 is given by v_1 v_2 v_3 v_4 v_1 and f_2 by v_2 v_6 v_4 v_5 v_2, p_1 is v_2, and p_2 is v_4. This gives a quick test for type II intersections: visit each vertex in turn and see if any of the virtual faces containing that vertex intersect.

Suppose that f_1 and f_2 are two virtual faces having a type II intersection; introduce the line segments of intersection as new edges, called *cutting edges*. Also introduce all the necessary points of intersection. The new vertices and edges are marked to distinguish them from the original vertices and edges. In general, these new vertices and edges will partition some of the virtual faces into smaller virtual faces. Using the algorithms described earlier, all those cutting edges which are bridges in a particular virtual face having type II intersections are identified. All virtual faces which induce these bridges are dropped, since they cannot possibly separate solid matter from empty space. Of course, after dropping some virtual faces, some of the type II intersections may disappear.

Since type II intersections are mostly the result of symmetry, we consider one of the most symmetrical cases possible, that shown in Fig. 6.6. After the regular octahedron of Fig. 6.6 passes through Stage 3, 11 faces will have been found: the usual 8 faces plus the 3 given by v_1 v_2 v_3 v_4 v_1, v_2 v_5 v_4 v_6 v_2, and v_1 v_5 v_3 v_6 v_1. The last 3 virtual faces (pairwise) have type II intersections. Each of the last 3 faces partitions each of the others into smaller virtual faces, which are all kept, ending up with 7 vertices, 18 edges, and 20 faces. The new wire frame is illustrated in Fig. 6.7.

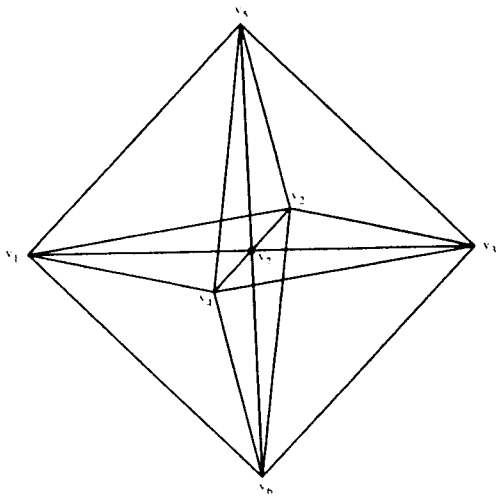


FIG. 6.7. A regular octahedrom after Stage 4 with cutting edges inserted.

Stage 5: Calculation of 2-Cycles and Virtual Blocks

In this stage virtual faces are fitted together to form candidate objects called virtual blocks. From the definition and discussion in Section 2, it is clear that objects can be found by calculating all 2-cycles and finding the nesting relationships among them. This 3-D process is a close analog of the 2-D process of fitting edges together to form virtual faces. However, the definition of a 2-cycle is in terms of $F(\mathbb{O})$, and at this stage of the algorithm only the virtual faces $VF(\mathbb{O})$ are available, where $F(\mathbb{O}) \subseteq VF(\mathbb{O})$. Thus, $VF(\mathbb{O})$ can contain elements which are not faces of $\mathbb{O}$ and are known as *pseudo-faces*. Pseudo-faces arise through chance alignments of edges and may occur in two forms:

I. The interior of the virtual face is empty space;
II. The interior of the virtual face is interior to solid material.

It will be seen that type I pseudo-faces are always rejected and that type II may either be rejected or be used to partition a primitive object into smaller subobjects.

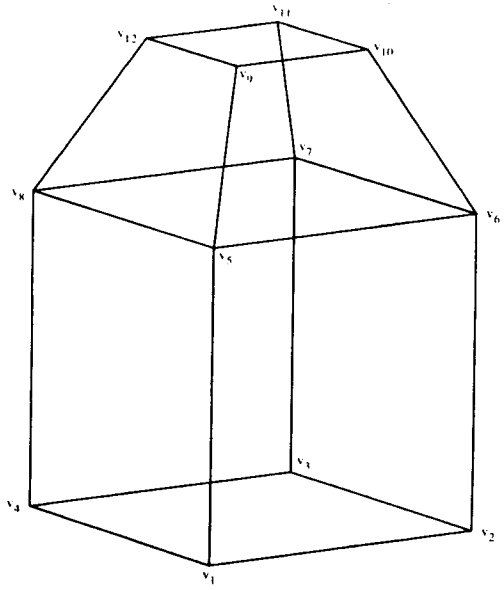
Some tests that detect pseudo-faces have been seen in Stage 4. An intersection of type I shows that the virtual face involved is really a pseudo-face. Similarly, an intersection of type II indicates that at least one of the virtual faces involved is a pseudo-face. Note that not every pseudo-face is involved in an illegal intersection of one of these two types. Another kind of pseudo-face that is detected in this stage is the 2-bridge, i.e., a virtual face which does not belong to any 2-cycle. After detecting and handling all of these pseudo-faces, the remaining

virtual faces naturally break up into 2-cycles. These 2-cycles partition all of $\mathbb{R}^3$ into connected components in much the same way that the 1-cycles partition the planes. In fact, the remainder of this stage is very similar to the virtual face creation algorithm of Stage 3. However, since this is in 3-dimensional space, no new types of intersections can occur and no new tests are necessary. As in Stage 3 some conventions are needed for describing the relationship beween virtual faces and the components of an object which they bound.

Let B_1 and B_2 be the two components of an object B^* whose boundaries (denoted by ∂B_1 and ∂B_2) both contain given a virtual face f. If the canonical normal (introduced in Stage 2) erected at any interior point of f points away from (into) B_1, then $+f \in \partial B_1$ ($-f \in \partial B_1$). Clearly, $+f \in \partial B_1$ ($-f \in \partial B_1$) iff $-f \in \partial B_2$ ($+f \in \partial B_2$). The goal is to find the various components of B^* because the original object can be built out of them.

Before proceeding further, consider a simple example. An object can have 1-cycles which result accidentally; in Fig. 6.8, the virtual face, $v_5\ v_6\ v_7\ v_8\ v_5$, is a pseudo-face, because it is not an actual boundary between empty space and solid material. However, this cannot be detected until the object is considered globally, i.e., when virtual faces are being found in the various planes, there is no way of distinguishing between faces and pseudo-faces. Only when the construction of the complete object in Fig. 6.8 is attempted is $v_5\ v_6\ v_7\ v_8\ v_5$ seen to be a pseudo-face.

This problem of pseudo-faces is handled by working with virtual blocks, i.e., 2-cycles which do not contain any nonbridge, virtual faces in their interior. Thus,



3. 6.8. An example of a pseudo-face.

virtual blocks are the primitive building blocks for dissection of an object by pseudo-faces. The object in Fig. 6.8 has three virtual blocks associated with it:

1. The closure of the unbounded component of B*;
2. The closure of the bounded component of B* lying above $v_5\ v_6\ v_7\ v_8\ v_5$;
3. The closure of the bounded component of B* lying below $v_5\ v_6\ v_7\ v_8\ v_5$.

To describe the boundaries in terms of the notation introduced above, assume that in Fig. 8 the origin is in the middle of the cube defined by v_1, v_2, v_3, v_4, v_5, v_6, v_7, and v_8, and that all plane positive normals radiate outward, giving

$$\partial B_1 = -f_1 - f_2 - f_3 - f_4 - f_5 - f_6 - f_7 - f_8 - f_9 - f_{11};$$
$$\partial B_2 = f_1 + f_2 + f_3 + f_4 + f_9 - f_{10};$$
$$\partial B_3 = f_5 + f_6 + f_7 + f_8 + f_{10} + f_{11};$$

where

f_1 is the face defined by $v_5\ v_6\ v_{10}\ v_9\ v_5$;
f_2 is the face defined by $v_6\ v_7\ v_{11}\ v_{10}\ v_6$;
f_3 is the face defined by $v_7\ v_8\ v_{12}\ v_{11}\ v_7$;
f_4 is the face defined by $v_8\ v_5\ v_9\ v_{12}\ v_8$;
f_5 is the face defined by $v_1\ v_2\ v_6\ v_5\ v_1$;
f_6 is the face defined by $v_2\ v_3\ v_7\ v_{6\ v2}$;
f_7 is the face defined by $v_3\ v_4\ v_8\ v_7\ v_3$;
f_8 is the face defined by $v_4\ v_1\ v_5\ v_8\ v_4$;
f_9 is the face defined by $v_9\ v_{10}\ v_{11}\ v_{12}\ v_9$;
f_{10} is the face defined by $v_5\ v_6\ v_7\ v_8\ v_5$;
f_{11} is the face defined by $v_1\ v_2\ v_3\ v_4\ v_1$.

Note that just as there were (in) and (out) 1-cycles, there are (in) and (out) 2-cycles. In the case above, ∂B_1 is an (out) 2-cycle, while ∂B_2 and ∂B_3 are both (in) cycles. As the reader probably suspects at this point, a tree of 2-cycles, similar to the one for 1-cycles, is constructed for an object. In fact, all the rules given for constructing a 1-cycle tree hold for 2-cycle trees. Virtual blocks are derived from this tree in the same way that virtual faces were derived from the 1-cycle tree. In this case the tree is represented in Fig. 6.9.

Before describing the procedure for finding 2-cycles, consider the case of the regular octahedron (Figs. 8.6 and 8.7). Because the octahedron is so symmetrical, it has three pseudo-faces each of which intersects the other two. Since there is no a priori method to eliminate any of them, either all possibilities can be tried or cutting edges can be introduced. Thus the octahedron of Fig. 6.7 decomposes into nine 2-cycles—one (out) and eight (in) 2-cycles.

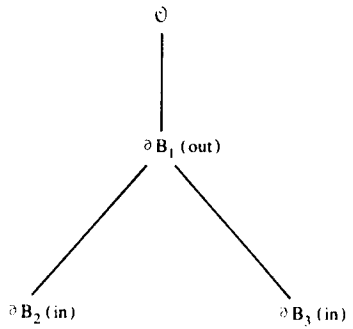


FIG. 6.9. Tree of 2-cycles.

The algorithmic steps to discover all 2-cycles and 2-bridges are now described. For each edge e of $\mathbb{O}$, a circular list is formed of all virtual faces which have e contained in their boundary. The faces are ordered in the same way as the corresponding edges were ordered in Stage 2, that is they are ordered radially around the edge. The search for 2-cycles now proceeds very much like the search for 1-cycles. Pick a virtual face with an orientation, i.e., $+f$ or $-f$, and attempt to find a virtual block containing it. Process edges one at a time by adding the appropriate face with the correct orientation, and maintain information on the number of times the edge is used and the sense of each use. Choosing an orientation for a virtual face is equivalent to assuming that solid material lies on a particular side of the virtual face. Thus an edge is processed by seeing which oriented faces contain it and picking those oriented faces which are neighbors through the solid material. Figure 6.10 illustrates this point by giving an edge-on view of the process. Suppose that the virtual faces f_1, f_3, f_4, and f_6 have been selected to be in 2-cycles with the orientations suggested in the figure by the normals and the shading. Since f_3 and f_4 are neighbors through solid material, they can both be dropped from further consideration. To find a virtual block both f_2 and f_5 would need to be added to the proposed 2-cycle with the indicated orientation. Note that it would be impossible for f_7 to belong to the 2-cycle because each edge can only belong to an even number of faces. If a virtual face is found which would need to be incorporated into the same 2-cycle twice (it will turn out that it is with opposite orientations), then that virtual face is a bridge and is deleted from the list of virtual faces. The partial results are saved and the process continued until the 2-cycle is completed. At the end of this process all bridges have been eliminated and every remaining virtual face belongs to exactly two distinct virtual blocks. Furthermore, the interiors of the virtual blocks are exactly the components of the complement of the remaining virtual faces. The original object must be a union of some of these virtual blocks, thus showing that in principle the problem has been reduced to a problem which involves only a finite number of possibilities. The next stage handles this last problem efficiently.

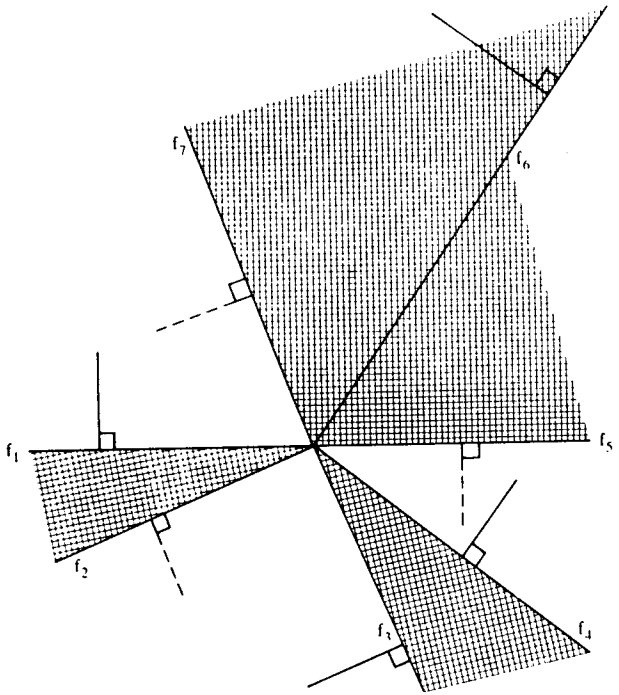


FIG. 6.10. Finding virtual blocks—an edge's perspective.

It only remains to mention one complication which can arise. In some cases, several edges of $\mathbb{O}$ are collinear and can be combined into a single line segment. In this case it is possible for one face to have as an edge a line segment which contains edges from other faces as subedges. In this case, there are a number of straightforward modifications which must be made to the 2-cycle finding algorithm.

Stage 6: Constructing all Solutions for the Wire Frame

In this stage virtual blocks are fitted together to generate all objects with a given wire frame. Basically each virtual block may have *solid* or *hole* state and, when a state assignment has been made to each virtual block, an object is obtained. However, not all assignments of solid and hole yield the desired wire frame. An assignment of solid or hole to the virtual blocks yields an object with the correct wire frame if

1. *Every* element $e \in E(\mathbb{O})$ belongs to two noncoplanar virtual faces f_1 and f_2 each of which belongs to one virtual block assigned solid state and one assigned hole state;

2. *No* cutting edge belongs to two noncoplanar virtual faces f_1 and f_2 each of which belongs to one virtual block assigned solid state and one assigned hole state, i.e., every cutting edge must be inside material.

A decision tree is constructed by growing those edges having the smallest number of unassigned virtual blocks containing them. The unique infinite virtual block is always assigned the hole state. Condition (1) is not always used to make choices between states; the necessary condition that every edge belong to a solid block and to a hole block is also used. However, conditions (1) and (2) are the ones that must be satisfied. To illustrate this process consider the regular octahedron of Fig. 6.7.

There are nine blocks:

B_1—the infinite virtual block;
B_2—the virtual block determined by $v_2\ v_1\ v_7\ v_5$;
B_3—the virtual block determined by $v_1\ v_4\ v_7\ v_5$;
B_4—the virtual block determined by $v_4\ v_3\ v_7\ v_5$;
B_5—the virtual block determined by $v_3\ v_2\ v_7\ v_5$;
B_6—the virtual block determined by $v_2\ v_1\ v_7\ v_6$;
B_7—the virtual block determined by $v_1\ v_4\ v_7\ v_6$;
B_8—the virtual block determined by $v_4\ v_3\ v_7\ v_6$;
B_9—the virtual block determined by $v_3\ v_2\ v_7\ v_6$.

Each edge of $\mathbb{O}$ now belongs to two virtual blocks of undetermined status while each cutting edge belongs to four virtual blocks of undetermined status. The state hole is assigned to B_1. An edge is picked, say $e = \overline{v_1\ v_5}$, and the decision tree begun.

Note that e already belongs to a block with hole state, so solid state must be assigned to some block. Figure 6.11 shows the decision tree in this case. Notice that each time the state of one of the B_i ($i \geq 2$) is set to hole a contradiction is quickly found. If B_2 has hole state, then B_3 and B_5 must be given solid state because $\overline{v_1\ v_5}$ and $\overline{v_2\ v_5}$ must belong to at least one solid block and there is only one candidate for this. However, if B_Z has hole state, and B_3 and B_5 solid state, the faces $v_1\ v_5\ v_7\ v_1$ and $v_2\ v_5\ v_7\ v_2$ contradict condition (2) for edge $\overline{v_5\ v_7}$. Similar contradictions arise whenever any B_i ($i \geq 2$) is treated as being empty. Notice that after a few assignments the subsequent choices are determined and exponential growth of the tree is avoided.

In some cases, there is an exponential number of different objects having the same wire frame, so exponential growth cannot be entirely avoided. However, if the tree is grown for depth, some object can be found having the given wire frame. In practice, this stage is completed fairly quickly since the geometry generally takes over once several assignments have been made. In complex

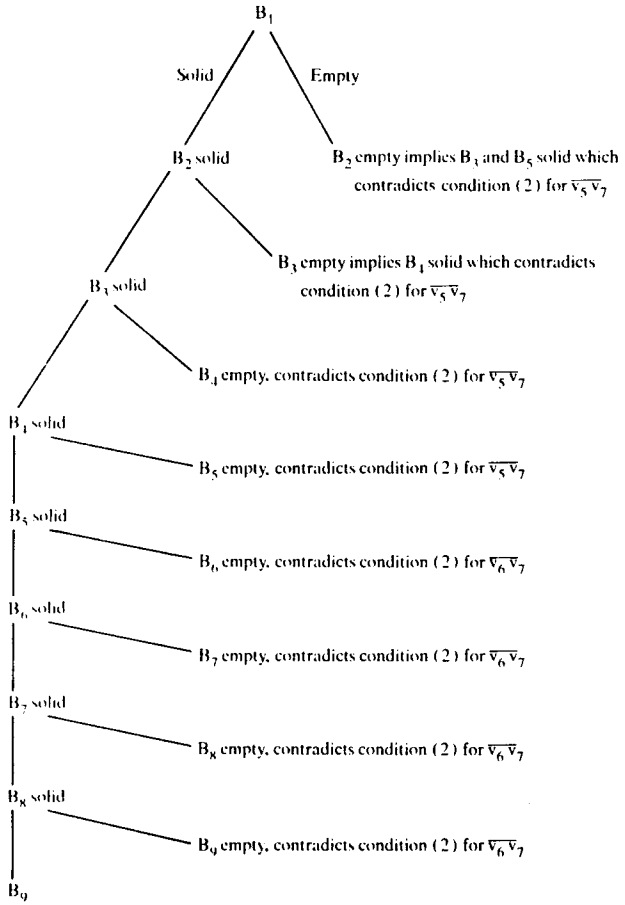


FIG. 6.11. A decision tree for the regular octahedron.

objects it is often the case that many edges on the outer boundary belong to exactly one virtual block which can be marked solid. In particular, any vertex belonging to exactly three elements of E(©) belongs to exactly two virtual blocks. Thus if one of them is empty, the other one must be solid.

Stage 6 feeds into an output module which puts the output together in forms which can be understood by the user of the system. The following section shows a number of examples in detail.

4. EXAMPLES

In this section are described a number of examples chosen to illustrate particular features of the algorithm. The examples are illustrated in Table 6.1.

Table 6.1(a) shows a double tetrahedron. Seven triangular virtual faces are found—the six outside faces and the internal area bounded by the waist of the figure. Three virtual blocks are found; the decision process assigns solid state to (1) and (2); block (3) is the unbounded virtual block; (1) and (2) are combined to produce the output object.

Table 6.1(b) shows an object with 1-D bridges on the faces containing abcd and kmnp. The plane graphs contain three bridges ef, kl, and op, none of which appear in the virtual faces for the planes shown. Two virtual blocks are found, one the output object and one the unbounded virtual block.

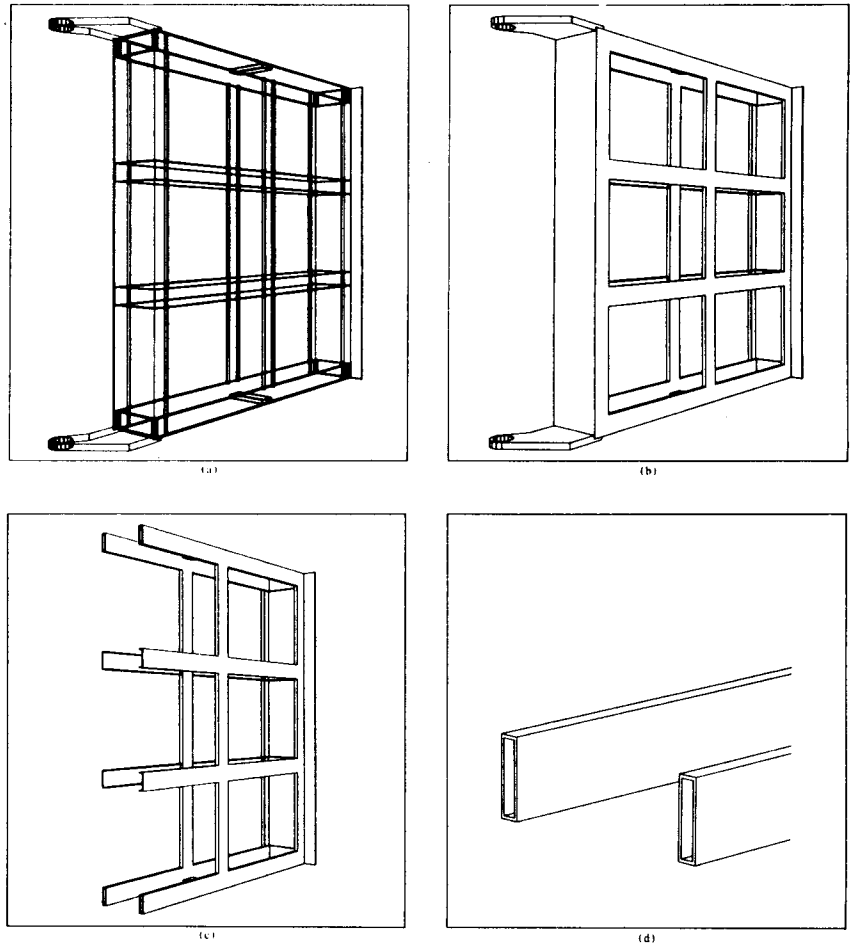


FIG. 6.12. (a) Wire frame with 1256 edges and 836 vertices; (b) volumetric representation of wire frame; (c) cross section of volumetric representation; and (d) close-up view of tubular members.

TABLE 6.1
Examples of the Wire Frame Algorithm

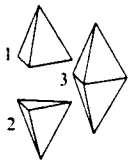
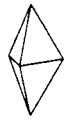
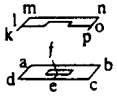
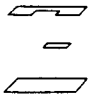
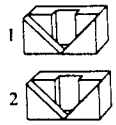
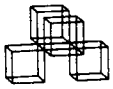
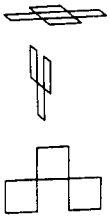
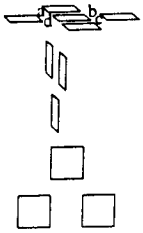
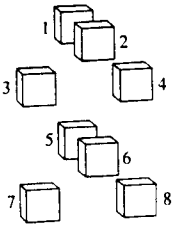
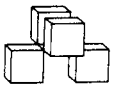
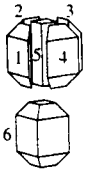
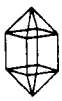
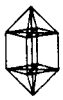
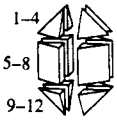
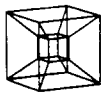
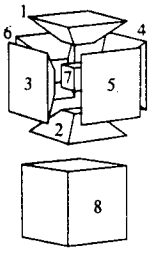
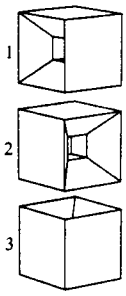
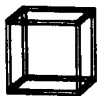
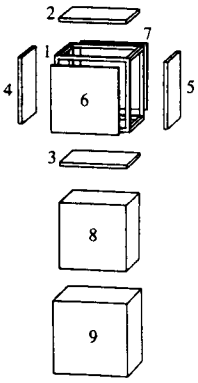
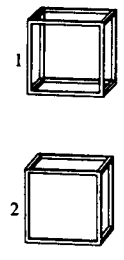
	Wire frame	Plane graphs of interest	Virtual faces of interest	Virtual blocks	Virtual block decision states (s = solid, h = hole)	Object
(a) Double tetrahedron		—			1 = s 2 = s 3 = h	
(b) 1-D bridges					1 = s 2 = h	
(c) 2-D bridges					1 = s 2 = s 3 = s 4 = s 5 = h 6 = h 7 = h 8 = h	
(d) Type I intersections			abcd and efgh have type I intersections and are not virtual faces		1 = s 2 = s 3 = s 4 = s 5 = h 6 = h	
(e) Type II interesections		—			1–12 = s 13 = h	
			With cutting edges inserted			
(f) Ambiguity I		—	—		1 = s s h 2 = s s h 3 = h s s 4 = h s s 5 = s h s 6 = s h s 7 = h h h 8 = h h h	
(g) Ambiguity II		—	—		1 = s s 2 = h h 3 = h h 4 = h h 5 = h h 6 = h h 7 = h h 8 = h s 9 = h h	

Table 6.1(c) shows four cubes positioned on two levels with four shared vertices enclosing a rectangular area abcd; abcd is found to be a virtual face, but in the virtual block building process is detected to be a 2-D bridge (i.e.,it is assigned opposite directions in the same virtual block to become a zero thickness sheet) and is not used in the output objects.

Table 6.1(d) contains an octahedron extended by a cube and pierced by a vertical square prism. The two plane graphs containing abcd and efgh have type I intersections with the vertical sides of the hole and therefore are not virtual faces. Six virtual blocks are found and assigned states as shown.

Table 6.1(e) shows the object of Table 6.1(d) without the piercing hole. Four face graphs with type II intersections occur and are shown as virtual faces with cutting edges inserted. Thirteen virtual blocks are found and assigned states as shown.

Table 6.1(f) shows a well known ambiguous wire frame (Voelker & Requicha, 1977); eight virtual blocks are found, and the decision process enumerates three valid solutions: one pair of opposing blocks (Sutherland, 1979; Boyse, 1979), (Wesley et al., 1980; Brown, 1977), or (Taylor, 1976; Woo, 1975) must have hole state, the center block (7) always has hole state.

Table 6.1(g) shows another ambiguous wire frame that could well occur in practice. Nine virtual blocks are formed; the decision process finds that block (8) can have hole or solid state.

Figure 6.12(a) shows a more complicated wire frame with 1256 edges and 836 vertices. In the course of the reconstruction process the wire frame algorithm finds 93 virtual blocks, most of them being window holes and enclosed volumes inside tubular members of the structure, and generates the volumetric representation shown in Fig. 6.12(b). Figures 6.12(c) and (d) show a cross section of the reconstruction with the nested interiors of tubular members correctly represented.

ACKNOWLEDGMENTS

Our thanks are due to D. D. Grossman for encouraging us to tackle the wire frame reconstruction problem and for providing the ambiguous example of Table 6.1(g), and to T. Lozano-Pérez for his contributions to early discussions on the problem.

At the time this work was done, the authors were located at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598. G. Markowsky is now at the University of Main, Orono, ME.

APPENDIX A: TOPOLOGICAL CONCEPTS

A brief introduction is given to those standard topological concepts used in this paper. For more details, see Hocking and Young (1961).

Definition A.1

Let $x \in \mathbb{R}^3$ and r be a positive number. $B_r(x)$ is used to denote the set of all points of $\mathbb{R}^3$ whose Euclidean distance from x is less than r. $B_r(x)$ is called the *open ball at x of radius r*. □

Definition A.2

A subset $X \subseteq \mathbb{R}^3$ is said to be *open* if, for all $x \in X$, there exists $r > 0$ such that $B_r(x) \subseteq X$. A subset $Y \subseteq \mathbb{R}^3$ is said to be *closed* if $\mathbb{R}^3 - Y$ is open. Note that open balls are open and that $\emptyset$ and $\mathbb{R}^3$ are both open and closed. □

Definition A.3

Let $X \subseteq Y \subseteq \mathbb{R}^3$. Then X is said to be *open in Y in the relative (induced) topology* [or open in the relative (induced) topology for short] if, for all $x \in X$, there exists $r > 0$ such that $B_r(x) \cap X = B_r(x) \cap Y$. X is *closed in the relative topology* if $Y - X$ is relatively open. □

In the cases most of interest here, i.e., subsets of a plane in $\mathbb{R}^3$, being relatively open means containing open disks (the intersection of a plane and an open ball). The following definitions will be stated only for the standard topology of $\mathbb{R}^3$ (Definition A.2) and the reader should verify that they make sense for any relative topology.

Definition A.4

The *closure*, $\bar{X}$, of a subset X of $\mathbb{R}^3$ is the set $\{x \in \mathbb{R}^3 \mid \text{for all } r > 0, B_r(x) \cap X \neq \emptyset\}$. In particular, $X \subseteq \bar{X}$. □

It can be shown that $\bar{X}$ is a closed set and that a subset $Y \subseteq \mathbb{R}^3$ is closed if and only if $Y = \bar{Y}$.

Definition A.5

The *boundary*, ∂X, of a set $X \subseteq \mathbb{R}^3$ is the set $\bar{X} \cap (\mathbb{R}^3 - X)$. □

Thus a point, x, is in ∂X if and only if there are points of both X and $\mathbb{R}^3 - X$ arbitrarily close to x.

Definition A.6

A subset X of $\mathbb{R}^3$ is said to be *connected* if two nonempty open subsets U_1, U_2 of $\mathbb{R}^3$ cannot be found such that $U_1 \cap U_2 = \emptyset$, $U_1 \cap X \neq \emptyset \neq U_2 \cap X$, and $X \subseteq U_1 \cup U_2$. □

In the case of $\mathbb{R}^3$ and its subplanes, all connected open subsets have the property that any two points in a given subset can be connected by a path which lies entirely in the given set.

Definition A.7

Let $X \subseteq \mathbb{R}^3$. A *connected component* of X is a subset Y of X which is connected and such that for any other connected subset $Z \subseteq \mathbb{R}^3$, either $Z \subseteq Y$ or $Z \cap Y = \emptyset$. □

Any set in $\mathbb{R}^3$ can be written as the disjoint union of its components.

Definition A.8

A subset X of $\mathbb{R}^3$ is said to be *bounded* (unbounded) if there exists for all $r > 0$ a point $p \in \mathbb{R}^3$ such that $X \subseteq B_r(p)$ [$X \not\subseteq B_r(p)$].

REFERENCES

Baer, A., Eastman, C. & Henrion, M. (1979, September). Geometric Modeling: A Survey. *Computer Aided Design, 11*, 253–272.

Boyse, J. (1979, January). Interference detection among solids and surfaces. *Commun. ACM, 22*, 3–9.

Brown, B. E. (1977, June). *Modeling of solids for three-dimensional finite element analysis.* Doctoral dissertation, Department of Computer Sciences, University of Utah, Salt Lake City.

Clowes, M. B. (1971). On seeing things. *Artificial Intelligence, 2*, 79–116.

Hocking, J. G. & Young, G. S. (1961). *Topology*. Reading, MA: Addition-Wesley.

Huffman, D. A. (1971). Impossible objects as nonsense sentences. In B. Meltzer & D. Michie, (Eds.), *Machine intelligence* (Vol. 6, pp. 295–324). Edinburgh University Press, Scotland.

Ikesawa, M. (1973, February). A system to generate a solid figure from a three view. *Bull JSME, 16*, 216–225.

Idesawa, M., Soma, T., Goto, E., & Shibata, S. (1975). Automatic input of line drawing and generation of solid figure from three-view data. *Proceedings of the International Joint Computer Symposium*, pp. 304–311.

Lafue, G. (1976, July). Recognition of three-dimensional objects from orthographic views. *Proceedings 3rd Annual Conference on Computer Graphics, Interactive Techniques, and Image Processing*. ACM/SIGGRAPH, pp. 103–108.

Lozano-Pérez, T., & Wesley, M. A. (1979, October). An algorithm for planning collision-free paths among polyhedral objects. *Commun. ACM, 22*, 560–570.

Requicha, A. G., & Tilove, R. B. (1978, March). *Mathematical foundations of constructive solid geometry: General topology of closed regular sets.* (Tech. Memo. No. 27). Production Automation Project. University of Rochester, New York.

Sutherland, I. E. (1963). SKETCHPAD: A man-machine graphical communication system. *Proc. SJCC, 23*, pp. 329.

Taylor, R. H. (1976, July). A synthesis on manipulation control programs from task level specifications. *Report No. STAN-CS-76-560* Stanford Artificial Intelligence Laboratory). Computer Science Department, Stanford University, Palo Alto, CA.

Udupa, S. (1977). *Collision detection and avoidance in computer controlled manipulators.* Doctoral thesis, California Institute of Technology, Pasadena.

Voelker, H. P., & Requicha, A. A. G. (1977, December). Geometric modeling of mechanical parts and processes. *Computer, 10*, 48–57.

Waltz, D. (1975). Understanding line drawings of scenes with shadows. In P. H. Winston (Ed.), *The psychology of computer vision* (pp. 19–91). New York: McGraw-Hill.

Wesley, M. A., Lozano-Pérez, T., Lieberman, L. I., Lavin, M. A., & Grossman, D. D. (1980, January). A geometric modeling system for automated mechanical assembly. *IBM J. Res. Develop., 24*, 64–74.

Woo, T. C.-H. (1977). *Computer understanding of design.* Doctoral thesis, University of Illinois at Urbana-Champaign.