Robust Regularized Set Operations on Polyhedra‡

Beat Bruderlin
Computer Science Department
University of Utah
Salt Lake City, UT 84112,
bruderlin@cs.utah.edu
Technical Report Number: UUCS-90-004

April, 1990

‡This work was supported in part by NSF grant # DDM-8910229

Robust Regularized Set Operations on Polyhedra‡

Beat D. Bruderlin
Computer Science Department
University of Utah
e-mail: bdb@cs.utah.edu

Abstract

This paper describes a provably correct and robust implementation of regularized set operations on polyhedral objects. Although the algorithm described here does not assume manifold polyhedra and handles all possible degenerate cases, it turns out to be quite simple. The geometric operations and relations are computed with floating point arithmetic which is efficient but not necessarily precise. To ensure that the results are still consistent we implemented a test that detects when dependent decisions contradict each other. The consistency test is relatively simple and can be carried out locally without having to reason about the logical dependencies of the geometric relations. The logical structure and the efficiency of the algorithm are barely influenced by the consistency test which makes this approach well suited for interactive modeling systems on modern workstations.

1. Introduction

Set operations are a powerful tool in geometric modeling for constructing solid objects from simple primitives. Geometric objects are regarded as point sets in the Euclidean space. New objects can be defined as the union, intersection, or set-difference of two such objects. Implementations of set operation algorithms can be found, for instance, in [BiNe 88], [Blo 86], [Bru 86], [Char 85], [FCM 87], [Hoff 89b], [HHK 89], [LBTH 86], [MaSu 82], [Mant 86], [Req 77], [Req, 80], [ReVo 85], [ThNa 87]

The first half of the paper mathematically specifies the algorithm for set operations in a way that makes the implementation provably correct, also for all degenerate cases, when we assume exact

‡ This work was supported in part by NSF grant # DDM-8910229

arithmetic. The algorithm finds the boundary representation from a Boolean expression over half spaces. The half space representation, chosen as the underlying data structure for this implementation has some advantages over the usual boundary representation: It describes solid objects without redundancy, and it facilitates the mathematical description of regularized polyhedra. Set operations handling degenerate cases are very complicated to implement otherwise.

The second half of the paper describes an efficient way of treating the problem of possible ambiguities, which are due to the inaccuracies of the underlying numerical representation and operations. The implementation is based on floating point operations. Therefore, degenerate cases (e.g. that two lines are parallel, or an edge is coplanar with a face, etc.) need to be decided with some tolerance. Such decisions are ambiguous and therefore the result can be inconsistent. In the case of set operations such ambiguities sometimes lead to unconnected edges and faces and other topologically inconsistent objects. Other approaches propose arbitrary precision rational arithmetic, or apply reasoning about dependencies of degeneracy decisions to make the result consistent. Those methods are generally more time consuming.

Our implementation is based on the optimistic assumption that, if the error estimations are done correctly, such inconsistencies occur rarely. We implemented a test that detects when a decision for a degenerate case is inconsistent with previous decisions. The algorithm can tell when the decisions are inconsistent, and fix the problem by adapting the tolerance. The approach is well suited for interactive geometric modeling, since it takes advantage of the floating point accelerators of modern workstations. The ambiguity test can be added to almost all existing efficient computational geometry algorithms without changing their logical structure. The algorithm is slowed down only by an insignificant constant factor in the majority of cases, namely, when no inconsistencies occur.

The paper also describes our experience with an interactive geometric modeling system based on this approach, which is implemented on Sun Workstations and Macintoshes.

2. Representing 3-dimensional Polyhedra

The implementation of set operations must reflect the data representation of the geometric objects. The most commonly used representation in CAD systems is the boundary representation:

2.1 Boundary Representation for Polyhedra

A body is defined by its boundaries, a set of facets which in turn are bounded by a set of edges (line segments) bounded by two vertices each (for an example see fig. 1).

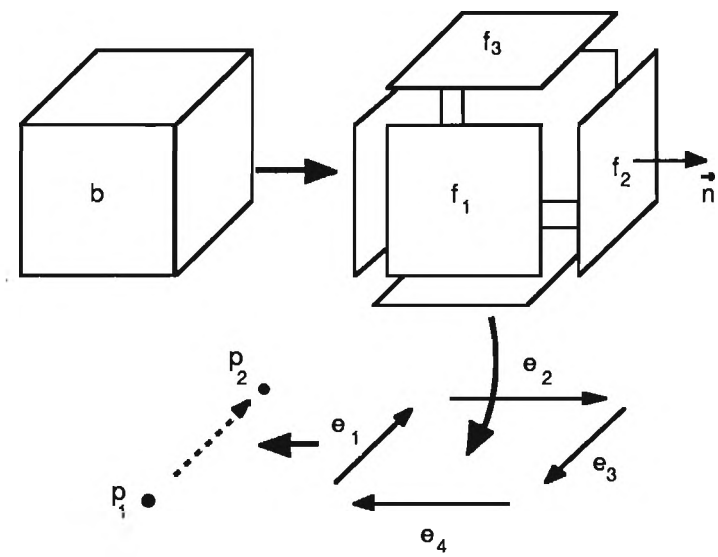


Fig. 1 The boundaries of a cube.

A polyhedron represented in boundary representation may be syntactically defined by the grammar in fig. 2.

```
body := { facet }
facet := "<" edge [{ "," edge }] ">"
edge := "<" vertex "," vertex ">"
vertex := "<" real "," real "," real ">"
```

Fig. 2 The EBNF-notation of a grammar for polyhedra in boundary representation.

The boundary representation is so popular because it directly supports many applications. For example, the edges and vertices are drawn, if an object is displayed on a graphical screen. If a hidden line drawing or a shaded image is to be drawn also the facets are important. For polyhedra represented this way the complete metric information is implicitly given by the coordinates of the vertices and may be derived using the syntactical definition. An edge may be interpreted as a vector going from one end-point to the other. The sequence of vertices indicates the direction of the vector. The sequence of edges in a facet is used to derive the surface normal vector as the sum of cross products of pairs of subsequent edge vectors. Algorithms for calculating volume properties, such as centroid, or moment-of-inertia, directly working on boundary representation can be found in [MeTa 80].

However, there are some inherent problems with boundary representation. The syntactical definition of the data structure alone does not guarantee that the polyhedron is semantically consistent; additional conditions have to be satisfied. Euler operators can be applied to generate topologically correct objects, as it has been done in [MaSu 82]. To guarantee that an object in boundary representation is topologically as well as geometrically consistent the definition of so called regularized polyhedra, as proposed in [Req 77], and [Req 80], is helpful in most practical applications. In addition to the syntactical definition we need to state semantic rules for regularized polyhedra. The following is an incomplete list:

a) All vertices belonging to the same facet must be coplanar. [1)]
b) The edges that belong to the same facet create cycles that must be closed and the cycle may not be self-intersecting.
c) Each edge belongs to exactly two facets (for two -manifolds).
d) The normal vectors of the facets must consistently look outward from the body.
e) All facets must close a body (no dangling facets).

[1)] This is an example of a rule that becomes necessary because of the data redundancy of the boundary representation.

f) A body must be closed on all sides.

With these rules we restrict the domain of possible polyhedra, and thus eliminate inconsistent definitions. Boolean operations, if directly carried out on boundary representation become very complicated in the case of degeneracies. Neighborhood information needs to be evaluated to disambiguate the various possible degenerate cases that might occur during the computation (see, e.g. [Hoff 89b], [ReVo 85]). In the following we want to find a data structure that disallows inconsistent data definition without the requirement of additional rules. We first define a regularized Boolean algebra which avoids dangling edges and faces (section 2.2). The half-space data structure described in section 2.3 avoids redundancies, and guarantees semantically correct data. We then apply the regularized Boolean algebra to the half space representation and thus obtain a definition of regularized polyhedra. In chapter 3 we mathematically relate the half-space representation to the boundary representation which leads, very naturally, to an algorithm for converting regularized Boolean expressions over half spaces into boundary representation.

2.2 A Regularized Boolean Algebra for Polyhedra

The Boolean algebra operates on the topological space T (the powerset of $\mathfrak{R}^n$) which is closed under the operations $\vee$ (union), $\wedge$ (intersection), $\neg$ (complement), ' (interior), and - (closure).

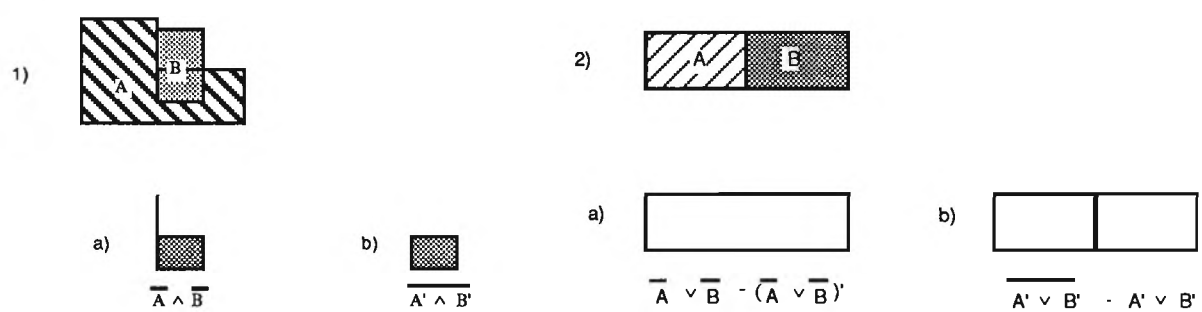


Fig. 3 Set operations on polyhedra with coplanar facets (cross section).

In the first example shown in fig. 3 (1) the intersection of two bodies is calculated. In case (a) a dangling facet is created which violates the requirements of regularized polyhedra, whereas case (b) gives an intuitively correct result. In the second example, shown in figure 3 (2) (the boundary of a

union of two bodies) in case (b) an undesired dividing facet is created. In the following we will define a consistent Boolean algebra that does not yield such undesired dangling edges and dividing facets.

We define a regularized Boolean algebra $\mathbb{B}^*$ operating on T^*, the set of all closed, regularized objects of T by.

$$T^* = \left\{ \bar{x} \mid x \subseteq \Re^n \wedge \overline{x'} = x \right\} \subset T, \quad \text{and} \quad \overline{\{\}} = \{\}$$

$$\mathbb{B}^* = \langle T^*, \wedge^*, \vee^*, \neg^* \rangle$$

Where the operations are defined as follows:

1) The *regularized complement* $\neg^*$ is defined by

$$\neg^* \bar{x} = \overline{\neg x} \in T^*$$

2) The *regularized union* $\vee^*$ is defined by

$$\bar{z} = \bar{x} \vee^* \bar{y} = \bar{x} \vee \bar{y}, \qquad \bar{z} \in T^*$$

3) The *regularized intersection* $\wedge^*$ is defined by

$$\bar{x} \wedge^* \bar{y} = \overline{x' \wedge y'} \qquad \in T^*$$

4) In addition to these basic operations we define the *regularized set difference* $-^*$ by

$$\bar{x} -^* \bar{y} = \bar{x} \wedge^* \neg^* \bar{y}$$

$$= \bar{x} \wedge^* \overline{\neg y} = \overline{x' \wedge (\neg y)'}$$

By verifying the axioms of Boolean Algebra for the regularized set operations it can be shown easily that $\mathbb{B}^*$ is a Boolean algebra. It is obvious that dangling edges and facets have an empty interior in an n-dimensional topological space, and therefore cannot be part of any object of T^*.

2.3 Representing Polyhedra by Half-spaces

Definition: A *half-space* is a subdivision of space by an oriented, infinite hyper-plane F^o into three subsets, F^o (the hyper plane itself), a positive (interior) side F^+, and a negative (outer) side F (fig. 4).

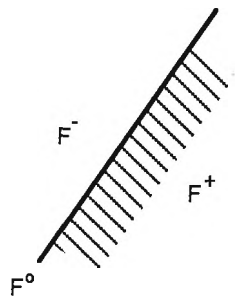


Fig. 4. A half-space.

In an n-dimensional vector space of $\Re^n$ half-spaces are represented by infinite sets of n-tuples (vectors):

$$\vec{x} = \langle x_1, \ldots, x_n \rangle$$

by the following linear forms:

$$F^o = \left\{ \vec{x} : \sum \lambda_i x_i = c \right\} \qquad c, \lambda_i \in \Re$$

$$F^+ = \left\{ \vec{x} : \sum \lambda_i x_i > c \right\}$$

$$F^- = \left\{ \vec{x} : \sum \lambda_i x_i < c \right\}$$

Definition: (Polyhedra, convex Polyhedra) A half-space defines a *primitive convex polyhedron* P. F^+ is the *interior*, of P, $F^+ \vee F^o$ is the *closure* of P, and F^- is the *complement*, $\neg$ P. By combining half-spaces with set operations we may define arbitrary polyhedra. A cube, for instance, may be defined as the intersection of six half spaces. In general, a polyhedron P, is a Boolean

expression on half-spaces. It is well known that every such expression can be written in disjunctive normal form:

$$P = \bigcup_{j} \bigcap_{k(j)} F_{j,k}^{\sigma_{j,k}} \qquad \sigma_{j,k} \in \{0, +, -\}$$

An intersection of half-spaces is convex. Therefore, each polyhedron in disjunctive normal form is the union of convex polyhedra.

Our next goal is to find an algorithm for transforming the Boolean expression of regularized polyhedra to boundary representation. For representing a regularized polyhedron P* we use the definition of regularized union, intersection, and difference introduced above. Again, we write P* in disjunctive normal form:

$$P^* = \bigcup_{j}^{*} \bigcap_{k(j)}^{*} F_{j,k}^{\sigma_{j,k}} = \bigcup_{j} \overline{\bigcap_{k(j)} (F_{j,k}^{\sigma_{j,k}})'}$$

$$= \bigcup_{j} \overline{\bigcap_{k(j)} F_{j,k}^{\pm}}$$

Note: A regularized polyhedron after the above definition, is closed but not necessarily finite. For a comprehensive treatment of the theory polyhedra, see, e.g. [Nef 78].

3 An Algorithm for Evaluating the Boundaries of Regularized Polyhedra

Regularized polyhedra can be expressed very elegantly as a mathematical formula over half spaces, as we saw above. In contrast to the boundary representation where the global, spatial information of an object is implicit and must be derived from (local) vertex coordinates and the topological information, the half-space representation defines the body explicitly. On the other hand, the elements of the boundary (vertices, edges, and facets) are implicit in the Boolean expression on half-spaces. Since most important algorithms in modeling systems are based on boundary representation, a transformation from a half-space representation to boundary representation is necessary. We define the elements of boundary representation in terms of half-spaces. We do this first for convex regularized polyhedra, and then use this definition for computing the boundary of

non-convex regularized polyhedra. The program finds the vertices, edges, and facets of regularized polyhedra which are given in half-space representation in disjunctive normal form. The most important parts of the program are coded in a Pascal-like programming language.

3.1 The Boundary of a Convex Polyhedron

A regularized convex polyhedron can be written as the intersection of half-spaces.

$$\overline{P} = \overline{\bigcap_j F_j^+}$$

The boundary of this polyhedron is its closure minus its interior:

$$b(\overline{P}) = \overline{P} - P' = \overline{P} \wedge \neg P'$$

$$= \overline{\bigcap_j F_j^+} \wedge \neg \bigcap_k F_k^+$$

$$= \overline{\bigcap_j F_j^+} \wedge \bigcup_k \neg (F_k^+)$$

$$= \overline{\bigcap_j F_j^+} \wedge \bigcup_k F_k^- \vee F_k^o$$

$$= \bigcup_k \underbrace{\left[F_k^- \wedge \overline{\bigcap_j F_j^+} \right]}_{\| \; \{\}} \vee \bigcup_k \left[F_k^o \wedge \overline{\bigcap_j F_j^+} \right]$$

therefore:

$$b(\overline{P}) = \bigcup_k F_k^o \wedge \overline{\bigcap_j F_j^+}$$

To obtain correspondence between polyhedra defined in boundary representation and their half-space representation we make the following definitions:

Definition: *d-dimensional faces (body, facets, edges, vertices).* We identify subsets of the boundary of a polyhedron P in $\Re^n$, and call them the d-dimensional "faces" ($0 \le d \le n$) of P. The only n-dimensional face of an n-dimensional polyhedron is the closed polyhedron itself. A d-dimensional face is the intersection of the boundary of a d+1 dimensional face with a d-dimensional hyper-plane, the interior of which is not empty in a d-dimensional topological space. Specifically we speak of the vertices as the zero-dimensional faces, the edges are the one-dimensional faces, and the facets are the two-dimensional faces. In the following, we restrict ourselves to three-dimensional polyhedra where the body is the three-dimensional face of the polyhedron. The zero-, one-, and two-dimensional faces of a regularized, convex polyhedron can be directly derived from the above formula (as subsets of the boundary).

Definition: The *d-dimensional faces* $0 \le d \le 3$

0-dim: *(vertex)* $\quad p_{i,j,k} = F_i^o \wedge F_j^o \wedge F_k^o \wedge \overline{\bigcap_l F_l^+}$

1-dim: *(edge)* $\quad e_{j,k} = F_j^o \wedge F_k^o \wedge \overline{\bigcap_l F_l^+}$

2-dim: *(facet)* $\quad f_k = F_k^o \wedge \overline{\bigcap_l F_l^+}$

3-dim: *(body)* $\quad b = \overline{\bigcap_l F_l^+}$

Lemma: For $d = 1..3$ the union of all (d-1)-dimensional faces, intersected with one d-dimensional face is equal to the boundary of this d-dimensional face.

Proof: For $d = 3$ (only one 3-dimensional face exists) the proof follows immediately from the above definitions and the computation of $b(\overline{P})$. For all lower dimensions the proof is left to the reader.

An algorithm to transform the half-space representation of a three dimensional convex polyhedron P in space into boundary representation has to find all zero-, one- and two-dimensional faces by evaluating the above Boolean expressions, and then associating them with the higher dimensional face for which it is part of the boundary. For example, for the vertex $p_{i,j,k}$ this means intersecting the three planes F_i^o, F_j^o, and F_k^o, by solving a system of three linear equations, and checking whether the resulting vertex, if it exists, is inside the closure of all half-spaces F_l^+ of the body. $p_{i,j,k}$ will belong to the boundary of $e_{i,j}$, $e_{i,k}$ and $e_{j,k}$. The geometric operations carried out are based on relations between two, three, or four half-spaces. In the following we discuss all possible cases, and classify them with respect to the possible geometric degeneracies. When the spatial relationship of two half-spaces is computed by the function spatial_relation(F1, F2) the following 7 cases are identified (figs. 5 and 6):

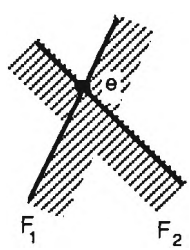


Fig. 5. Generic case: F_1^o and F_2^o intersect.

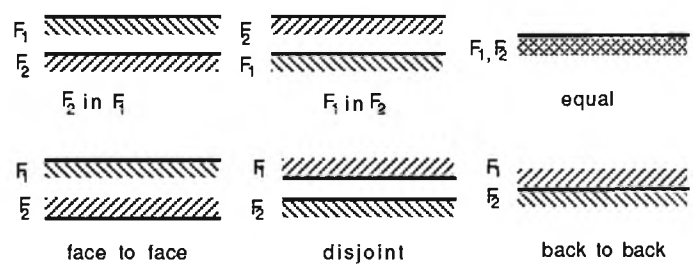


Fig. 6. Degenerate cases: F_1^o and F_2^o are parallel.

In case the planes F_1^o and F_2^o intersect an infinite intersection line e is computed by the function Line(F1, F2). The procedure "Faces_of_convex", described below, finds the faces of convex polyhedra. The knowledge of the convexity of the body **B** is also used for deciding whether a half-space is redundant, or completely outside B, and may therefore be deleted, or whether B is the empty set, and no further evaluation is required. No overlapping facets or edges are created by the algorithm. In this algorithm a body is regarded as a set of facets. A facet is a set of edges. For

adding an element to, or for subtracting an element from a set, the two operations '+' and '-' are used respectively in the algorithm.

```
procedure Faces_of_convex(B: ConvexBody);
begin;
   for each F1(j) in B do
      for each F2(k) in B where (k < j) do
      case spatial_relation(F1(j), F2(k)) of
         intersecting :     Edge(B, e, F1(j), F2(k));
                            F1(j) := F1(j) + e;
                            F2(k) := F2(k) + e|
         F2_in_F1     :     B := B - F1(j)|
         F1_in_F2     :     B := B - F2(k)|
         equal        :     B := B - F1(j)| (* delete one of the two *)
         face_to_face :                     |
         disjoint     :     B := {}         |
         back_to_back :     B := {}         |
         end-;
      od;
   od;
end Faces_of_convex;
```

The intersection line **e** is computed in the procedure 'Edge' which is described below. The line **e** has to be intersected with all other half-spaces S of the body. Either **e** and the corresponding planes form an intersection such that we may compute an end-point of the edge (fig. 7), or **e** is parallel to the plane, in which case a further evaluation of their spatial relation must be made (fig. 8). This evaluation is done by the function half-space_line_relation(F1, F2, S, angle).

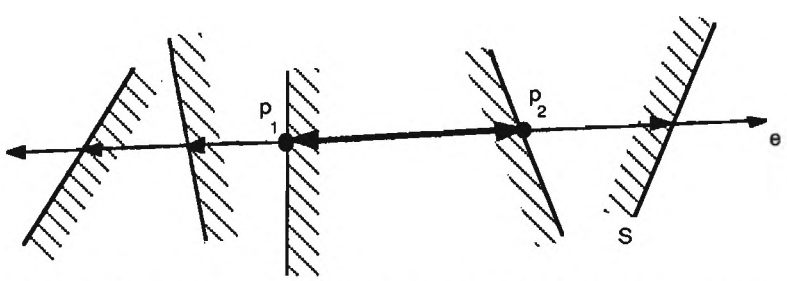


Fig. 7. Generic case: Intersecting **e** with other half-spaces.

The procedure Vertex(S,e) computes the intersection point of **e** and the Plane S^o, and if possible, adds it to **e**. For each edge **e** only the innermost two intersection vertices p_1 and p_2 remain as end-points (convexity). If the segment between the end-points is outside the polyhedron the edge

becomes obsolete. If **e** is parallel to the plane of **S** it may be on the plane, inside or outside the half-space (fig. 8).

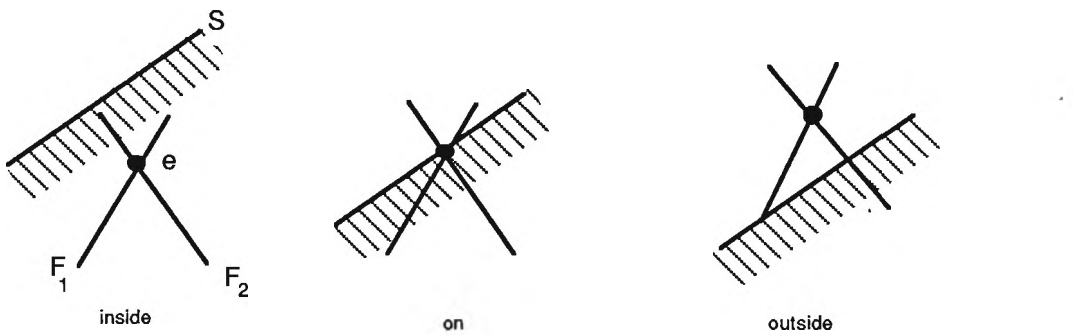


Fig. 8. Degenerate cases: Edge **e** is parallel to S (cross section perpendicular to e shown).

In this case also the angle of **S** relative to the space spanned by $F_1^+ \wedge F_2^+$ is calculated by the function half-space_line_relation(F1, F2, S, angle). The following eight cases are distinguished (fig. 9). A corresponding value [1 .. 8] is assigned to the variable *angle*.

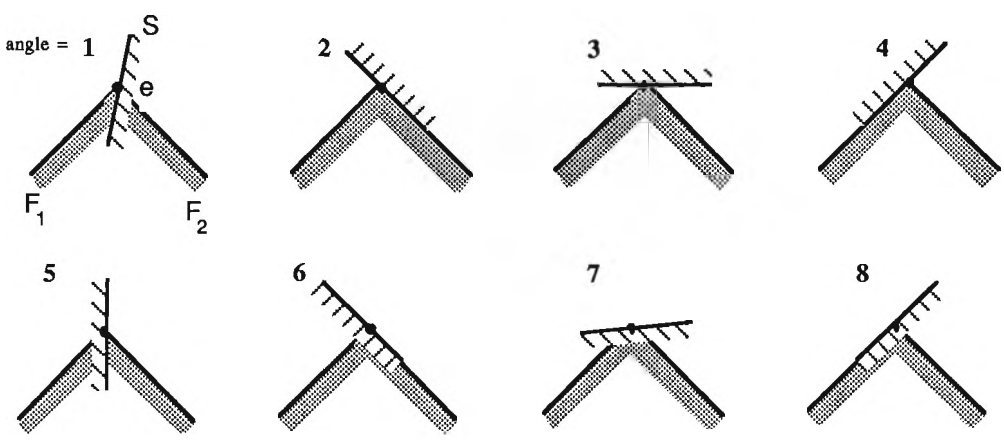


Fig. 9. Classifying the angle of S with respect to F_1 & F_2 (section perpendicular to **e** shown).

The procedure "Edge" finds the one-dimensional faces of a convex polyhedron.

```
procedure Edge(B: ConvexBody; F1,F2: Face): Edge;
begin;
   e := Line(F1, F2);
      for each (* facet *) S in B where (S ≠ F1 & S ≠ F2) do
```

```
        case half-space_line_relation(F1, F2, S, angle) of
           intersecting:
             Vertex(S, e)
           |
           on:
             case angle of
               2,3,4    :B := {}|
               6,7,8    :B := B - S |
               1        :e := {}; B := B - F1|
               5        :e := {}; B := B - F2 end;
           |
           inside:
             case angle of
               6,7,8    :B := B - S
             else end

           |
           outside:
             e := {}
             case angle of
               2,3,4    :B := {}|
               8,1      :B := B - F1|
               5,6      :B := B - F2|
             else end;
        else end;
     od;
     return e;
end Edge;
```

3.2 The Boundary of Non-convex Polyhedra

Non-convex regularized polyhedra in disjunctive normal form are expressed as the union of convex sub-polyhedra. Bringing a Boolean expression to disjunctive normal form, as is required by the algorithm, is a purely syntactical transformation of the data structure which is done in a pre-processing step of the program. The boundary of such a polyhedron is:

$$b(P^*) = \bigcup_j \overline{P}_j \wedge \neg \left[\bigcup_k \overline{P}_k \right]' \qquad \text{since } \forall P_j : \; P_j' \subseteq \left[\bigcup_k \overline{P}_k \right]' \qquad \text{we may write}$$

$$b(P^*) = \bigcup_j \left[\overline{P}_j \wedge \neg P_j' \right] \wedge \neg \left[\bigcup_k \overline{P}_k \right]'$$

$$b(P^*) = \bigcup_j b(\overline{P}_j) \wedge \neg \left[\bigcup_k \overline{P}_k \right]'$$

In other words, we can compute the boundary of a non-convex polyhedron by first evaluating the boundary of the convex sub-polyhedra to which we add their mutual intersection edges. We then subtract from each edge the part that intersects with the interior of the total polyhedron. An edge of a sub-polyhedron that is coplanar to a facet of another sub-polyhedron belongs to the boundary, but is meaningless if in the interior of this facet, and therefore will be discarded. This is done by simply counting the number of times the edge is incident with one of the facets of the other convex polyhedron.

The procedure 'Intersection_edges' computes the mutual intersection edges of two convex bodies B1 and B2, and adds them to both intersecting facets.

```
procedure Intersection_edges(B1, B2: ConvexBody);
   begin;
   for each (* facet *)  F(j) in B1 do
      for each (* facet *) F(k) in B2 do
         if spatial_relation(F(j), F(k)) = intersecting
         then
            e := Line(F(j), F(k));
            F(j) := F(j) + e;
            F(k) := F(k) + e;
            for each (* facet *) S in B1 ∨ B2 where (S ≠ F(j) & S ≠ F(k)) do
               case half-space_line_relation(F1, F2, S, angle) of

               intersecting : Vertex(S, e)
               |
               on:
                     (* e is collinear with another boundary edge
                       and is therefore no longer necessary in
                       the body to which S belongs *)

                  if S ∈ B1 then F(j) := F(j) - e end;
                  if S ∈ B2 then F(k) := F(k) - e end
               |
               inside:

               |
               outside: e:= {}
               end
            od
         elsif spatial_relation(F(j), F(k)) = equal
         then (* combine the edges of the two facets *)
               F(j) := F(j) ∨ F(k); F(k) := {}
         end
      od
   od
end Intersection_edges;
```

The procedure 'subtract_interior' computes the part of an edge e(j,k) that is inside the convex body B, and then subtracts it from e(j,k).

```
procedure subtract_interior(e(j,k): Edge; B: ConvexBody);
begin;
   subtractfrom1 := true;
   subtractfrom2 := true;
   segm := e(j,k);
   counter := 0 (* counts to how many planes e is coplanar
                   if coplanar to one -> inside a facet
                   if coplanar to two -> on boundary of facet *)
   for each (* facet *)  F(l) in B where l ≠ k & l ≠ j do
      case half-space_line_relation(F(j),F(k),F(l), angle) of
      intersecting   : Vertex(S, segm)
      |
      on:
         counter := counter + 1;
         case angle of
            6,7,8:|
            4,5   : subtractfrom2 := false | (* leave segment for F(k) *)
            1,2   : subtractfrom1 := false | (* leave segment for F(j) *)
            3     : subtractfrom1 := false; subtractfrom2 := false
         end
      |
      inside:
      |
      outside:
         segm := {}
      end;
   od;
   if counter < 2 then
     if subtractfrom1 then F(j) := F(j) - e(j,k);
                           F(j) := F(j) + diff(e(j,k), segm) end;
     if subtractfrom2 then F(k) := F(k) - e(j,k);
                           F(k) := F(k) + diff(e(j,k), segm) end;
   end;
end subtract_interior;
```

The boundary representation of a three-dimensional polyhedron is the sum of all its zero-, one-, and two-dimensional faces. The algorithm, as it is described so far, ensures that the d-dimensional faces do not intersect with the interior of a higher dimensional face for which they build the boundary. Faces of equal dimension, however, still may overlap with each other. Such overlapping faces are eliminated by:

- Merging coplanar facets to become one facet. The edges belonging to the same facet are combined into so called rings (cycles). If the sense of rotation of a ring is in the same direction as the surface normal vector of the facet, it is an exterior boundary of the facet. If it is in opposite direction, the ring is a surface void (see fig 10).
- Merging collinear edges. Since the facets have a direction represented by a normal-vector the intersection lines of two facets have the direction of the cross-product of the two normal vectors. Coincident edges are merged, if they have the same direction, or their overlapping part is erased, if oppositely directed (see fig. 11).
- Storing vertices with the same x-, y-, z- coordinates only once, since they are identical.

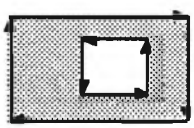


Fig. 10. Creating a surface void.

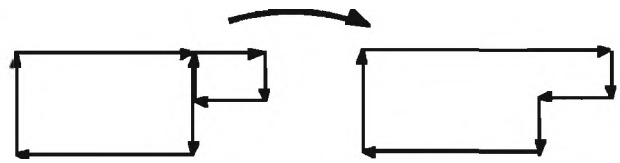


Fig 11. Merging collinear edges of coplanar facets.

4. Problems of Accuracy and Robustness

In the algorithm described in this paper we assumed that the computations and decisions about degeneracy conditions could be made with absolute precision. The representation of the coordinates by floating-point values on a computer is only an approximation of $\Re$. This means that relations like coincidence of vertices, parallelism and coincidence of lines and facets can only be decided within the accuracy of the representation of floating-point numbers, and the accuracy of the operations carried out on these objects. The correctness of the result depends on the fact that the decisions made in the algorithm about the relations are consistent. Due to the inaccuracies of the representation by floating point numbers and the arithmetic operations, such decisions are questionable. When, for instance, we want to decide whether two planes intersect or are parallel we can compute the angle between them. When the angle is larger than the error of computation we can be sure that the planes intersect. On the other hand, when this angle is smaller than the resolution of floating point numbers

it may either mean that the two planes in reality are parallel (i.e. the angle should be zero, but deviates from zero due to inaccuracies of previous computations), but it could also mean that the planes intersect in reality with a very small angle. We have no way to absolutely distinguish between the two possibilities. The approach taken by most modelers that use floating point operations is to arbitrarily decide that, when the angle α is smaller than some ε (representing the uncertainty of the data) α is assumed to be zero, otherwise $\alpha \neq$ zero. In most practical cases this approach turns out to work well. We have to choose ε large enough, such that the largest occurring error during computations is smaller than ε. However, we can always find, or deliberately generate situations, for which a decision following the above rule leads to a contradiction. When the angle between two planes is just around ε, at some instance the algorithm might decide that two planes are coplanar according to the ε criterion. In another, similar instance it decides that they intersect. When trying to connect the edges resulting from plane-plane intersections we may find that the continuation edge has not been generated because of this ε criterion. The decisions were made independently, although they may logically depend on each other. The result can be a topologically inconsistent object with dangling edges or faces. This problem has been addressed in a some recent papers. A general overview on the approaches taken can be found in [Hoff 89a], [Hoff 89b]. A heuristic reasoning approach to find out about the interdependencies of decisions in an program for set operations on polyhedra is described in [HHK 89]. This way, inconsistencies in the result can be avoided. So far, no approach that yields a provably correct result in all cases has been found. Also, the reasoning approach adds some additional overhead to the algorithm and often changes its asymptotic behavior.

In the approach taken here, we test for possible inconsistencies without reasoning about the logical interdependency of decisions. The test that detects ambiguities is done whenever a spatial relation between half-spaces is computed. We first describe the basic ideas at the example of point coincidence, and later extend to higher dimensional objects, such as lines and planes.

Each point is represented by a spherical ε region, defining an upper bound for the error. It is correct to say that two points are distinct when the ε regions don't overlap. However, there is no

criterion to determine when two points are coincident. When the ε regions overlap we have a certain probability that the two points are equal. When we arbitrarily decide that they are equal this might be inconsistent with other decisions. We can easily construct an example involving three points A, B, C, where the ε rule leads to an inconsistency. Let's assume the ε regions of points A and B overlap; also the ε region of B overlaps with the ε region of C, but the ε regions of A and C don't overlap. Therefore, points A and C cannot be equal. Deciding that A = B, and B = C would contradict the decision that A $\neq$ C. Therefore, at least one of the other pairs is not equal. The ε region does not help us with that decision. On the other hand, we want to handle degenerate cases, since they are often intentional and therefore very important. Very often, the ε criterion works successfully in detecting those degenerate cases. The mentioned example is somewhat unusual in that a definitely non-degenerate case (or small feature) A $\neq$ C came to lie close to a probably degenerate case A = B (where B $\neq$ C), or B = C and A $\neq$ B. We base our approach on the philosophy that deciding that points are incident whenever the epsilon regions overlap is corrects as long as it does not logically contradict other decisions. All we therefore want to do is to detect inconsistencies, when they occur. The method can be described as follows:

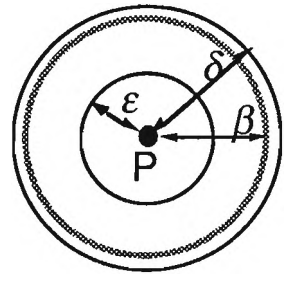
1) We define a second region δ around each point. δ completely contains ε. We say two points are distinct when the δ regions don't overlap.
2) Otherwise, when the δ regions overlap, and when the ε region of one point is contained in the δ region of the other point, we say that the two points are equal.

A consequence of saying that two points are equal is that we have to enlarge the ε region of the two points. The new ε region for both points will be the union of the two previous ε regions. This indicates that one point can be wherever the other point can be. This way, the two points "remember" the decision that they are equal to some other point. The role of the δ region is two-fold. First, it serves as a distinction region, in the sense described above. Second, it is an upper

bound to the growth of ε. When ε is no longer completely contained in δ, some decision made earlier is ambiguous.

We now describe the data structures used in the program. For each point we define three regions, ε, β and δ (see fig. 12). The meaning of the regions ε and δ are the ones described above. β initially is the same as δ. Whenever two points are considered equal, the new ε region of the two points will be the union of the two ε regions [1], and their common β region will become the intersection of the two previous β regions [2].

Fig. 12 Point with data with error
Tolerance region: ε
Buffer region: β
Distinction region: δ



Definitions (coincident, distinct, equivalence class, consistent set of points).

Two points P_j, P_k, are *coincident*: $P_j = P_k$, iff $\varepsilon_j \subseteq \delta_k \wedge \varepsilon_k \subseteq \delta_j$

Two points P_j, P_k, are *distinct*: $P_j \neq P_k$, iff $\delta_j \cap \delta_k = \varnothing$

The points P_1, .. P_k, build an *equivalence class* $\stackrel{k}{\underset{i=1}{=}} P_i$, iff $\bigcup_i \varepsilon_i \subseteq \bigcap_i \delta_i$

A set of point S is *consistent*, if $\forall P_j, P_k \in S$, either $P_j = P_k$, or $P_j \neq P_k$.

---

1) To make this operation geometrically simple we approximate the new uncertainty region ε of both points by the smallest sphere including the union of the two ε spheres.

2) For simplicity we approximate the new β region by the biggest sphere included in the intersection of the two β spheres.

This definition of consistency can actually be computed for the data structure of points. With the starting condition, $\forall P_i\ \varepsilon_i \subseteq \delta_i$, and the update operations upon pair wise comparison of points, if

we don't find an ambiguous situation, i.e. as long as $\varepsilon_i \subseteq \beta_i$, where $\beta_i = \bigcap_{P_j = P_i} \delta_j$.

Lemma 1 If a set of points S is consistent then coincidence of points is an equivalence relation for this set.

Proof of Lemma 1 We can easily show that from the above definitions follows:

$\forall P_j,P_k \in \left(= P_i\right) \subseteq S$: $P_j = P_k$, and $\neg\exists\ P_j,P_k \in \left(= P_i\right)$: $P_j \neq P_k$. Therefore, $P_j \neq P_k$ is equivalent to $\neg\ (P_j = P_k)$. The transitivity, reflexivity, and symmetry properties of the coincidence relation can be directly derived. □

Higher dimensional objects, such as lines and planes can be handled in a similar way as points. To make the comparisons and updates simple, their error regions should have simple shapes complementing the dimensionality of the object in the 3-dimensional space. They also should be symmetric in the complement space. The ε, β and δ regions of line segments in 3D therefore have the form of a cylinder (see fig. 13). The cylinders are bounded on both sides, such that all points incident with a line are within the two ends of the cylinder. The uncertainty regions of planes are planar plates with thickness ε, β or δ, respectively. Again, we limit the valid region of the plane such that it contains all line segments and points incident with that plane. A simple way of approximating this is to define a cylindrical disc with height ε, β or δ, and a radius large enough to contain all the points and lines incident with that plane (see fig. 14).

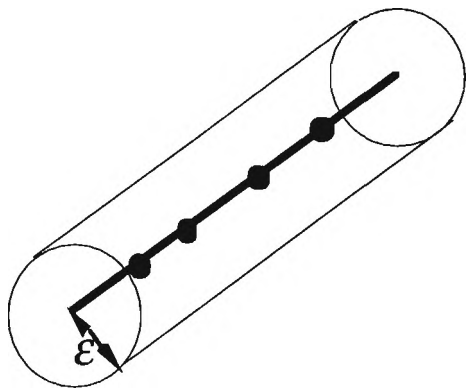


Fig. 13. The ε environment of a line in 3D.

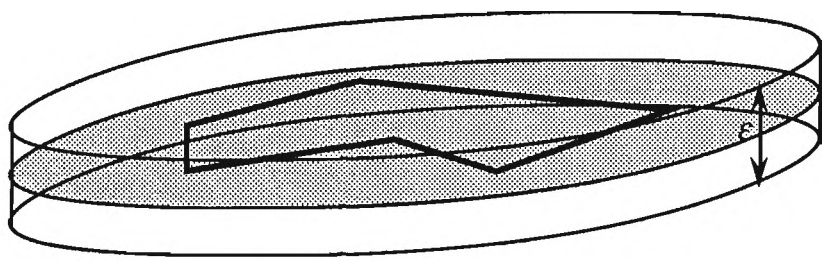


Fig. 14. The ε environment of a plane in 3D.

Since the algorithm described in the previous paragraph is based on half spaces which have an infinite extension, we have to estimate the finite extensions of the error regions by bounding boxes. In the interactive modeler, all objects are generated from finite primitives (cubes, prisms, etc.) for which we already know the extensions, therefore the extensions of the error regions can be approximated easily. Computing relations between points, lines, and planes and updating the ε, β and δ regions works similarly as for points. Basically, one has to carry out set operations on these regions. To avoid possibly complicated geometric operations the result is again approximated by spheres or cylinders. The following example shows how incidence of a point P and a plane F is decided consistently (fig. 15).

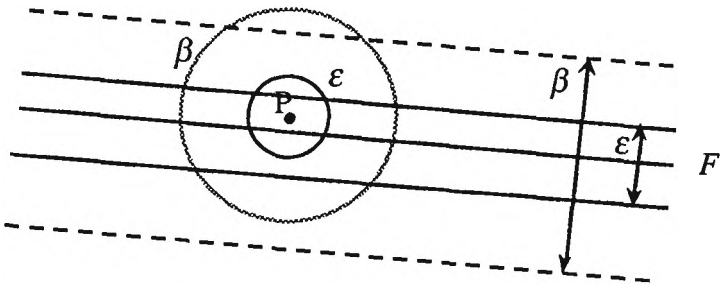


Fig. 15. Incidence of point P and plane F with ε and β regions (cross section).

The radius of the ε sphere of the point and the width of the ε disk of the plane are increased by the minimal amount, such that the bounding planes of the disk become tangent to the sphere. (see fig 16)

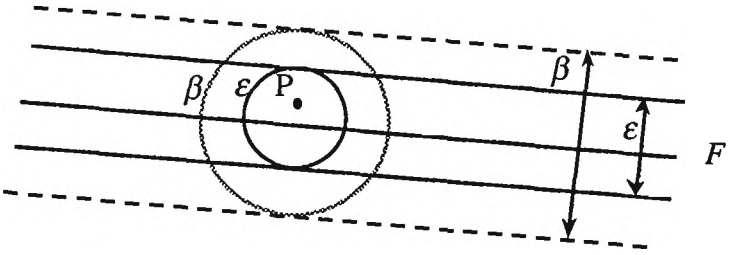


Fig. 16. The updated ε and β regions of the point and plane.

The radius of the β sphere of the point and the width of the β disk are decreased by the minimal amount such that the bounding planes of the disk become tangent to the β sphere. The diameters of the ε and β sphere of the point will be equal to the widths of the ε and β plates of the plane. We define that a plane and a point are incident, if the described updates can be made consistently. If the ε region is not completely contained in the β region after the update an ambiguity will be reported to the main algorithm.

Intersecting two planes F_1 and F_2 should have the same outcome as computing the incidence of a line with two non-coplanar planes. Updating ε and β is therefore handled accordingly (an

illustration is given in fig. 17). Note that the ε and β regions of the two planes will have equal width after intersecting.

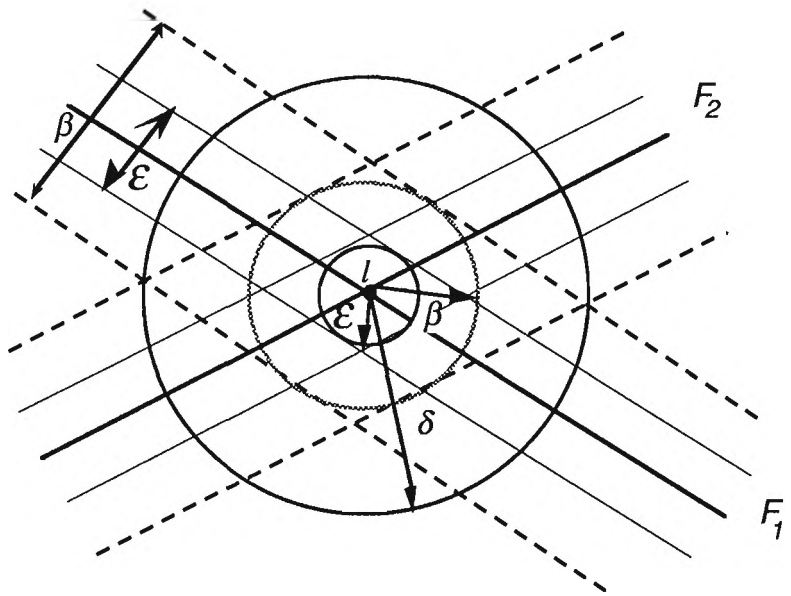


Fig. 17. Intersecting two planes F_1 and F_2 with ε and β regions yielding the intersection line l (cross section).

The δ region of the intersection line will have at least the maximal extent of the intersection of the two β regions of the planes. Thus, we take into consideration that intersecting two planes at a small angle is less accurate. In the extreme (when intersecting two coplanar planes) the δ region would be infinitely large. Since collinear planes are merged and not intersected this case will not occur.

The ambiguity test can be done in constant time, and therefore does not change the asymptotic time behavior of the algorithm. In most practical cases the distinction between degenerate and non-degenerate cases is unambiguous and no special measures have to be taken. The influence on the performance of the algorithm in these cases is insignificant. In the rare cases of ambiguities the test localizes the problem. The algorithm never returns an inconsistent object, instead it will return a message that the operations could not be computed with the given tolerance values. To cure the problem the program has to recompute the set operations with larger tolerance regions ε, and larger

upper bounds δ for all the points, lines and planes. The δ regions will be large enough to include the ε region updated during the previous run. More of the technical details of this method and a proof that, whenever the test doesn't find an ambiguity, then the results are consistent, are described in [Bru 90].

5. Conclusion

The algorithm described in this paper is part of a 3-D modeler, which has been implemented for Macintoshes and for SUN workstation, using the Suntools window system. The modeler is part of a project on constraint-based geometric modeling, described in [Bru 87]. The interactive user works with primitives such as cubes, and prisms, etc. which may be translated, rotated, and scaled, and applies Boolean operations on them. The user deals with geometric objects in the so called Constructive-Solid-Geometry-way (CSG). The resulting objects again can be treated like primitives, and further Boolean operations can be applied on them. This way, more complicated objects can be generated from simple ones. An example scenery, generated on a Macintosh, is shown in fig. 18. The example contains objects generated by set operations with a mixture of generic and degenerate cases. Internally the objects are represented by half-spaces.

The goal of this research was to develop an implementation of regularized set operations which is robust, easy to implement, and preserving the underlying mathematical principles. We first applied a regularized Boolean algebra to polyhedra represented by a Boolean expression over half-spaces. For this representation a correspondence to the elements of boundary representation (vertices, edges, facets) can be expressed mathematically. The geometric interpretation of the formulas directly leads to an algorithm for transforming a half-space representation of a polyhedron to its boundary representation.

The ambiguity test, as it is described in this paper has been added to the algorithm later. The only changes to the program that became necessary were a) to extend the data structures for points, lines and facets to also contain information on the ε, β and δ regions, and b) we had to re implement the geometric operations, such as, for instance, intersecting planes, and the computation of geometric

relations, such as incidence, etc. Experiments show that in non-degenerate cases the algorithm runs as fast as our previous not robust version. In the case of degeneracies, some additional overhead has to be done to update the ε, β and δ regions which slows down the algorithm by an average of 10% compared to the previous, not robust version [BRU 86]. In the rare cases where ambiguities are detected the whole set operation has to be run again with a higher tolerance. The small features that lead to ambiguities before will now be consistently treated as degenerate cases, but possibly new, somewhat bigger features will lead to ambiguities (this would require another run of the algorithm, etc.). In practice, it turns out that most cases are well behaved, and require one, or at most two runs in some rare cases.

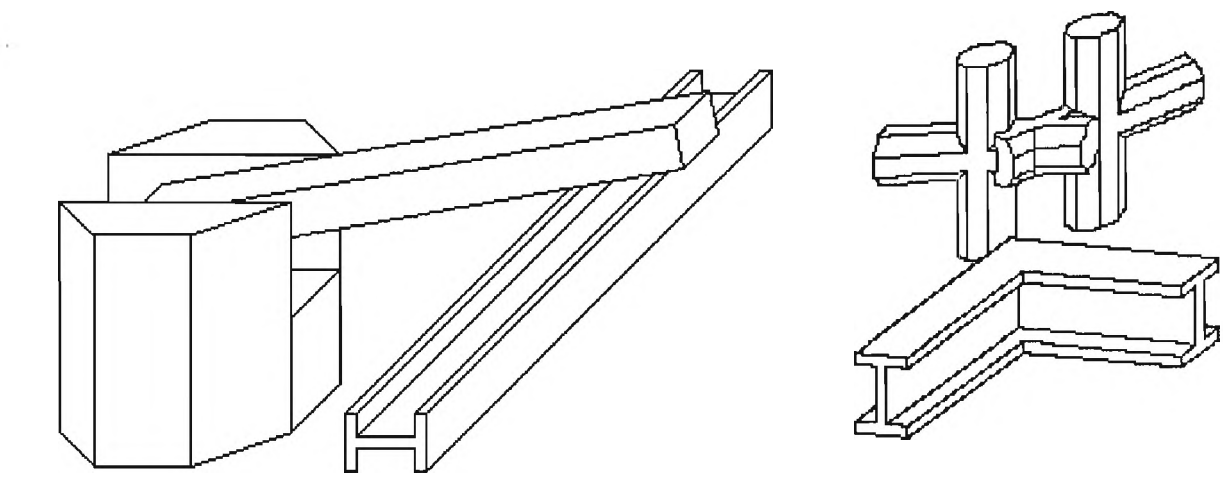


fig. 18. Example: Simple scenery containing generic and degenerate cases.

As a test case we took an example similar to the one described in [HHK 89]. We take two identical cubes and rotate one them slightly around all three coordinate axes by some angle α, and then carry out a union operation on the two cubes. With a floating point precision of 6 decimal digits we detect some ambiguities when the angle α is between 0.5 and 10^{-5} degrees. When α is smaller

than 10^{-5} degrees this is uniformly treated as a degenerate case and the union of the two cubes is equal to one of them. When α is larger than 0.5 degrees the algorithm clearly yields a non-degenerate case. In the case of an ambiguity the program automatically extends the ε, β and δ regions according to the previous findings and recomputes the Boolean operations. With the new tolerances this will yield a degenerate case, i.e. the two cubes are treated as equal.

Acknowledgements

I would like to express my gratitude to Jan Charvat who implemented the first version of the set operation algorithm, and to Shiaofen Fang who implemented the ambiguity check. This work was supported in part by NSF grant # DDM-8910229.

References

[BiNe 88] H. Bieri, W. Nef. Elementary Set Operations with d-Dimensional Polyhedra. In Computational Geometry and its applications. Lecture Notes in Computer Science, No. 333, Springer Verlag, 1988

[Blo 86] Sandra H. Bloomberg. A Representation of Solid Objects for Performing Boolean Operations. Technical Report No 86-006. The University of North Carolina at Chapel Hill. Department of Computer Science, 1986

[Bru 86] Beat Bruderlin. Regularized set operations on polyhedra. Tech report No 86-08, Mathematik, ETH Zurich, 1986.

[Bru 87] Beat Bruderlin. Rule-based Geometric Modeling. Ph.D. thesis, Computer Science, ETH Zurich, December 1987

[Bru 90] Beat Bruderlin. Detecting Ambiguities: An Optimistic Approach to Robustness and Accuracy Problems in Computational Geometry Algorithms. Tech. Report No UUCS-90-003, Computer Science, Univ. of Utah, April, 1990.

[Char 85] J. Charvat. Regularized set operations on polyhedra. M.S. thesis, ETH Zürich, 1985

[FCM 87] J. Flaquer, A Carbajal and M. A. Mendez. Edge-edge relationships in geometric modeling. Computer-Aided Design, Vol. 19, No. 5, Butterworth, June 1987

[Hoff 89a] C.M. Hoffmann. The problems of Accuracy and Robustness in Geometric Computation. IEEE Computer, Vol 22, No. 3, March 1989.

[Hoff 89b] C.M. Hoffmann. Geometric & Solid Modeling, An Introduction. Morgan Kaufmann Publishers Inc., 1989.

[HHK 89] C.M. Hoffmann, J.E. Hopcroft, M.S. Karasick. Robust Set Operations on Polyhedral Solids. IEEE Computer Graphics and Applications, November 1989

[LBTH 86] David H. Laidlaw, W. Benjamin Trumbore, John F. Hughes. Constructive Solid Geometry for Polyhedral Objects, ACM Siggraph '86, Conference Proceedings Dallas, August 18 - 22, 1986

[MaSu 82] M. Mäntylä, R. Sulonen. GWB: A Solid Modeler with Euler Operators. Computer Graphics and Applications, Vol. 2, No. 2, March 1982, pp. 1-97.

[Mant 86] M. Mäntylä. Boolean Operations of 2-Manifolds through Vertex Neighborhood Classification. ACT Transactions on Graphics, Vol. 5 No. 1, January, 1986.

[MeTa 80] A.M. Messner, G.Q. Taylor. Solid Polyhedron Measures, Collected algorithms from the ACM, alg. No. 550 (1980). ACM Transaction on Mathematical Software, Vol. 6, No. 1, March 1980, pp 121 - 130

[Nef 78] Walter Nef. Beiträge zur Theorie der Polyeder, mit Anwendungen in der Computergraphik, Herbert Lang, Bern, 1978.

[Req 77] A.A.G. Requicha. Mathematical models of rigid solid objects. Production Automation Project TM-28 University of Rochester.

[Req 80] A.A.G. Requicha. Representation for Rigid Solids: Theory, Methods, and Systems. Computing Surveys, Vol. 12, No.4, December 1980

[ReVo 85] A.A.G. Requicha, Herbert B. Voelcker. Boolean Operations in Solid Modeling: Boundary Evaluation and Merging Algorithms. Proceedings of the IEEE, Vol 73, No. 1, January, 1985.

[ThNa 87] William C. Thibault, Bruce F. Naylor. Set Operations on Polyhedra Using Binary Space Partition Trees. ACM Siggraph Conference Proceeding, Computer Graphics, Volume 21, Number 4, July, 1987.