Robust Boolean set operations for manifold solids
bounded by planar and natural quadric surfaces.

Xiaohong Zhu, Shiaofen Fang, Beat D. Bruderlin

UUCS-92-020

Department of Computer Science
University of Utah
Salt Lake City, UT 84112 USA

October 16, 1992

Abstract

This paper describes our latest effort in robust solid modeling. An algorithm for set operations on solids bounded by planar and natural quadric surfaces, that handles all geometrically degenerate cases robustly, is described. We identify as the main reason for the lack of robustness in geometric modeling, that dependent relations are handled inconsistently by disregarding the dependencies. Instead of using explicit reasoning to make dependent decisions consistent, we show that redundant computation can be avoided by correctly ordering the operations, and redundant data can be eliminated in the set operation algorithm, so that the result is guaranteed to be a valid two-manifold solid.

Robust Boolean set operations for manifold solids
bounded by planar and natural quadric surfaces.

Xiaohong Zhu, Shiaofen Fang, Beat D. Bruderlin
Computer Science Department
Salt Lake City, UT 84112

Abstract:

This paper describes our latest effort in robust solid modeling. An algorithm for set operations on solids bounded by planar and natural quadric surfaces, that handles all geometrically degenerate cases robustly, is described. We identify as the main reason for the lack of robustness in geometric modeling, that dependent relations are handled inconsistently by disregarding the dependencies. Instead of using explicit reasoning to make dependent decisions consistent, we show that redundant computation can be avoided by correctly ordering the operations, and redundant data can be eliminated in the set operation algorithm, so that the result is guaranteed to be a valid two-manifold solid.

1 Introduction

It is generally acknowledged that geometric algorithms often fail in some degenerate cases such as coplanar planes, co-cylindrical cylinders and cylinders tangent to planes, etc. These cases occur a lot in engineering design, and it is important that they are handled robustly.

Geometric objects are conceptually continuous, yet the computation we use is always discrete (for instance, floating point numbers are used to approximate real numbers). A geometric representation includes both geometric information and topological information. In a geometric algorithm, the topological information is often constructed by computing relations from the geometric information. Because these relations are often interdependent, an algorithm can easily create inconsistencies in the topological data, when the computation is not precise. Algorithms with inconsistent topological data will either fail or generate non-sense results as shown with some examples in [6] and [13].

A number of recent publications attempt to perform precise computations by using exact numbers [10][25][32][33]. The authors made the assumption that geometric shapes can be represented by exact numbers, which is true, only for a limited domain (e.g. polyhedra).

The approaches in [5] and [35] use geometric perturbations to avoid positional degeneracy. However, avoiding degeneracies is not always acceptable in geometric modeling because they are usually intentional. Applying symbolic reasoning to guarantee the consistency of the geometric relations has been done by some researchers[15][20][31]. However, due to the complexity of the symbolic reasoning problem, they can only solve problems with limited implications or incompletely.

Tolerance-based approaches have been studied in [1][2][7][8][6][29]. These methods use tolerances to keep certain information of the algorithm's decision-making history. When making a new decision, the algorithm will check the tolerances to make the decision consistent with all the previous ones, and update the tolerances to reflect the new decision. While the results of tolerance-based approaches are generally satisfactory, significant extra computation is needed for all the tolerance operations. Also, inconsistencies cannot be completely avoided but only detected, and then resolved, usually by rerunning parts of the algorithm.

We believe that robustness can be achieved for a specific algorithm by carefully ordering the primitive operations of the algorithm with no or very little extra computation. We observe that robustness problem is mainly caused by a contradiction of redundant and imprecise computation. Imprecise computation does not generate inconsistencies if there is no redundancy in the data representation. In this paper we present an approach of removing all the redundancies in a Boolean set operation algorithm for two-manifold 3D objects. The algorithm is robust, and at the same time efficient, because there is basically no extra computation required for achieving robustness.

2 Derivation of Robustness

To remove redundancies, we should first identify them, and then remove them properly.

2.1 What is a redundancy

We distinguish two types of redundancies:

1. *Direct redundant data computation.* For instance, when we intersect two planes f1 and f2, and later on intersect f2 with f1, this means that we compute the intersection of f1 and f2 twice, but possibly with a different result. We call the redundancy caused by this direct redundant data computation a *direct redundancy.* This type of redundancy

can be easily avoided. The idea of removing direct redundancy has been represented in [15].

2. *Indirect redundant data computation.*

   For three planes, denoted by f1, f2 and f3 (as shown in figure 1), we first compute the intersection line e of f1 and f2. After this, we compute the relation of line e and the plane f3. If we find line e is on the plane f3, then the following implications apply.

$$\left.\begin{array}{lll} e & = & f1 \cap f2 \\ e & \text{is on} & f3 \end{array}\right\} \Rightarrow \left\{\begin{array}{l} e = f1 \cap f3 \\ e = f2 \cap f3 \end{array}\right.$$

   The intersection of f1 and f3, and the intersection of f2 and f3 are both coincident with e. In other words, the decision that e is on f3, in fact, implicitly computes the intersection of f1 and f2, and f2 and f3. Explicitly computing the other intersections in other parts of an algorithm would therefore create redundant data. The enlargement in figure 1 shows that, due to numerical imprecision there are actually 3 intersection lines. Using a tolerance to decide the relations might determine that e is on f3 (the distance between e and f3 is smaller than that tolerance) but that the three intersections are not coincident (because their relative distance is slightly larger than the tolerance), which is contradictory, according to the above finding.

   We call the redundancy caused by such indirect, redundant data computation an *indirect redundancy.*

Totally removing redundancies in geometric algorithms would require geometric theorem proving, which is too complicated to be realized practically. In special cases, such as regularized boolean set operation, under the assumption that objects are two manifolds, we can attempt to eliminate redundancies.

2.2 A hybrid data representation

A hybrid data representation has been used in this algorithm to represent objects, which combines the *half-space representation* and *boundary representation* to obtain direct access to both boundary and volume information.

The half-space representation of a solid can be expressed in the following normal form:

$$\bigcup_i \bigcap_j F_{i,j}^{+}$$

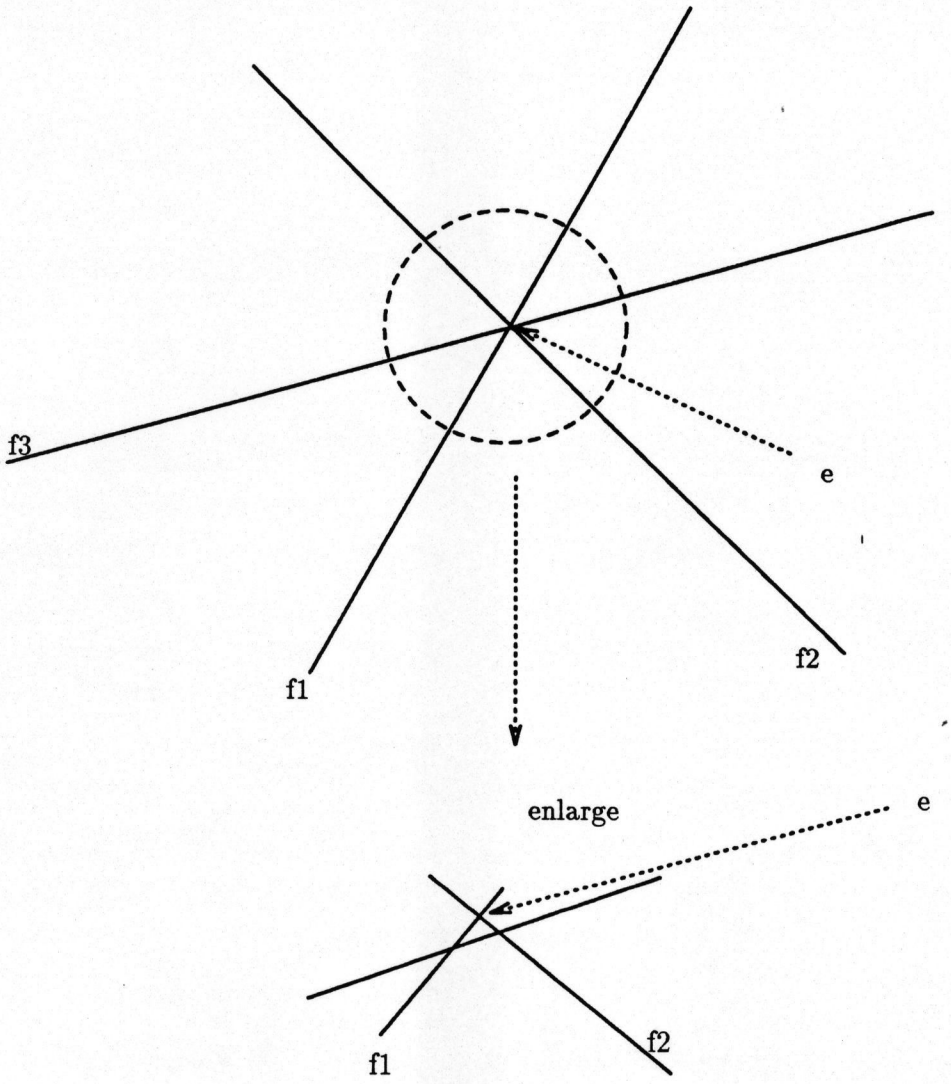

Figure 1: three surfaces intersecting in one line

where the union and intersection operations are regularized set operations [12]. And Fi,j is an implicit surface, represented by implicit equation:

$$F : f(x, y, z) = 0$$

F subdivides the 3D space into three regions: F^+, F^0 and F^-, namely the positive half space, the surface and the negative half space, where:

$$F^+ = \{p : f(p) > 0.0\}$$

$$F^0 = \{p : f(p) = 0.0\}$$

$$F^- = \{p : f(p) < 0.0\}$$

Each term $B_i = \bigcap_j F_{i,j}^+$ in the half space representation is called a generalized convex body, denoted as GCB. A GCB is convex when all its bounding half spaces are convex (which is the case for the planes and natural quadrics used in our implementation). In general, however, a GCB is not a 3D convex set because not all half spaces are convex. The half space representation of a solid can then be described as a union of GCBs.

The boundary representation has two parts, a topological description of the connectivity and orientation of vertices, edges, and faces, and a geometric description for embedding these elements in space[12].

The boundary representation we used in this algorithm is slightly different from the well known half-edge representation described in the literature. The object has a list of bounding faces. Each face is bounded by a set of disjointed edge cycles which we call rings. One of the rings is the outside boundary of the face, the others (if exist) are bounding holes. Each vertex is the intersection of two edges that are on the same face (rather than in 3D). Each edge contains the following information :

- incident vertices
- neighbor half-edge
- the face it belongs to
- successive edge
- direction of the edge (The convention is that the interior of the face is to the left, looking from outside)

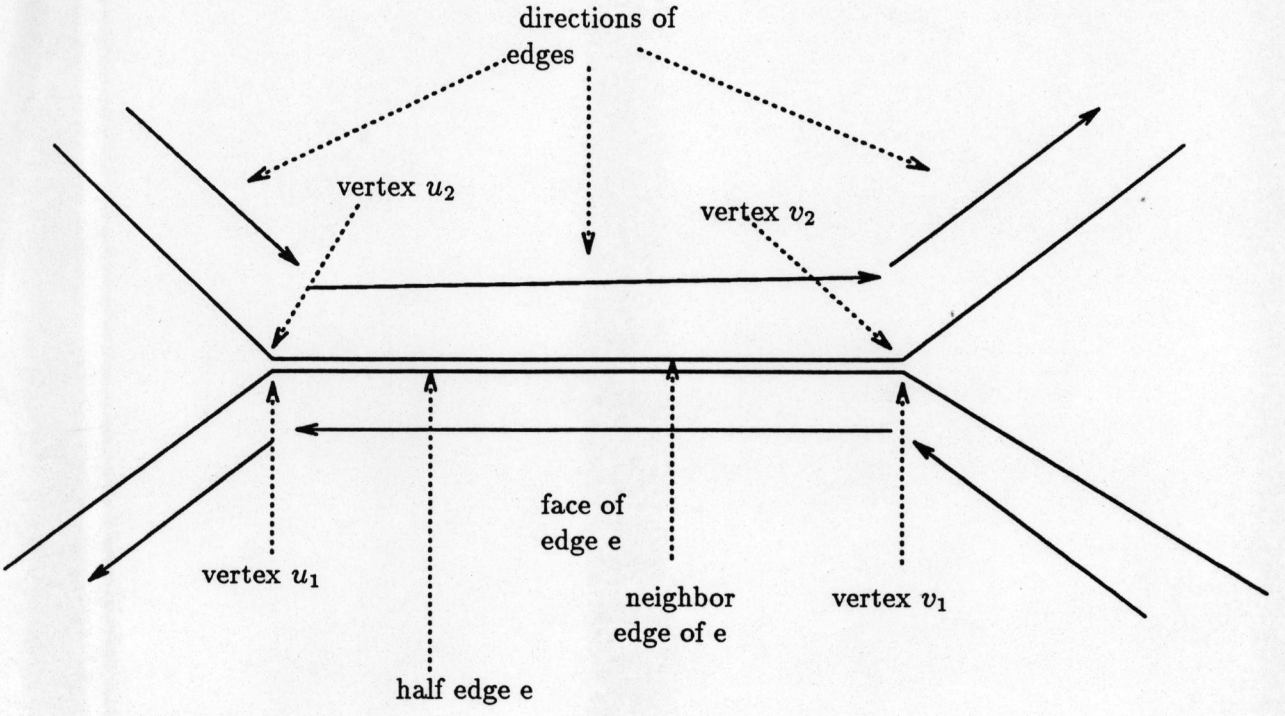

Figure 2: The boundary representation

The edge is oriented by distinguishing the two incident vertices by their order, one is defined as *from* vertex, and the other is defined as *to* vertex. The schema is shown in figure 2.

2.3 Regularized boolean operations on hybrid representation

Regularized boolean set operations are an important tool used in constructing objects in geometric modeling. The operations are *regularized union*; *regularized intersection*, and *regularized difference*. They differ from the corresponding set theoretic operations in that the result is the closure of the interior of the solid, which eliminates "dangling" edges and faces, and isolated vertices.

In the half space representation, 3D boolean operations are extremely simple. We only need the following set of rewrite rules for Boolean union (+), intersection (*) and difference (−).

$$(\bigcup_i \bigcap_j F^+_{i,j}) + (\bigcup_k \bigcap_l G^+_{k,l}) \Rightarrow (\bigcup_i \bigcap_j F^+_{i,j}) \cup (\bigcup_k \bigcap_l G^+_{k,l})$$

$$(\bigcup_i \bigcap_j F^+_{i,j}) * (\bigcup_k \bigcap_l G^+_{k,l}) \Rightarrow \bigcup_i \bigcup_k [(\bigcap_j F^+_{i,j}) \cap (\bigcap_l G^+_{k,l})]$$

$$(\bigcup_i \bigcap_j F^+_{i,j}) - (\bigcup_k \bigcap_l G^+_{k,l}) \Rightarrow (\bigcup_i \bigcap_j F^+_{i,j}) * \overline{\bigcup_k \bigcap_l G^+_{k,l}}$$

where an overline denotes the complement set.

The geometric data of the representations are half spaces. The potential redundancies of this representation are the coincident identical surfaces (Null objects [Tilove] are not considered redundant, but superfluous and will also be deleted; they don't affect the consistency).

For the boundary representation, Boolean operations are fairly involved. The basic steps are:

- Compute the membership classifications of the boundaries of one solid versus the other solid. This involves computing the intersection between boundaries of different solids and carrying out the inside/outside/on tests for a boundary against a solid.
- Collect boundaries of the new solid by selecting parts of the boundaries (depending on the Boolean operations) after the membership classification.
- Build the topological relationships and boundary hierarchy of the new solid.

From the above we can see, that there are two kinds of computations: "direct" intersection computation (face vs. face, or edge vs. edge), and "indirect" intersection computation by computing geometric relations (incidence, coincidence). The latter one may lead to redundancy and inconsistency.

2.4 Avoiding and Removing Redundancies

In a previous section, we analyzed redundancies occurring in regularized boolean set operations, next we want to either avoid the redundancy, or remove it when it is detected.

For the half space representation, to remove the redundancies, we will merge all the surfaces that are considered coincident within the specified tolerance. Afterwards, such identical half spaces may still occur in different places of the Boolean expression, but they will point to the same surface equation.

For the boundary representation, to remove the redundancies, we need to do the following operations:

For the direct intersection computation, including the face-face, edge-edge intersections, we use a similar approach, as mentioned in[15]. The direct redundancies, only depend on the algorithm, not on the geometry or relations, therefore, it is easy to rearrange the order of the algorithm to avoid these redundancies.

For the indirect intersection computation (incidence), we cannot avoid all the redundancies ahead of time. Fortunately, we can detect redundant data, and delete it, when necessary.

Relations between edges and faces can be intersecting, parallel, and on.

The incidence relation of an edge with a face means that three faces intersect in one common edge. If we can assume that the object resulting from a Boolean set operation is a 2 manifold, then we know that each edge must be the intersection of *exactly* two faces. However, temporarily we may obtain such an, *incident edge-face relation leading to a redundancy that we need to detect and remove* (during the computation of set operations such incidence relations occur relatively often).

For example, as shown in figure 3, e is the intersection edge of face f1 and face f2. Face f1, and face f2 are faces of the same GCB c1, and face f3 belongs to GCB c2. We determine that e is on the face f3. From the topological relations we can decide that in the union of the two GCBs the final boundary edge e will be deleted and replaced by the intersection edge of f1 and f3, and f2 should not intersect f3 and f1. Because edge e is the intersection edge of f1 and f2 it will be deleted from both surfaces.

Vertices, in our approach are computed by intersecting two edges that belong to the same face. Each edge is the intersection of 2 surfaces, but one of the surfaces is shared by the two intersecting edges, therefore, a vertex is incident to exactly 3 surfaces.

Similarly to above, we assume that each face has 1-manifold topology. Therefore, we can apply the same rule, namely that a vertex is only a intersection of two edges, and if it is incident with a third edge in that face we can delete part of the redundant data and ensure consistency.

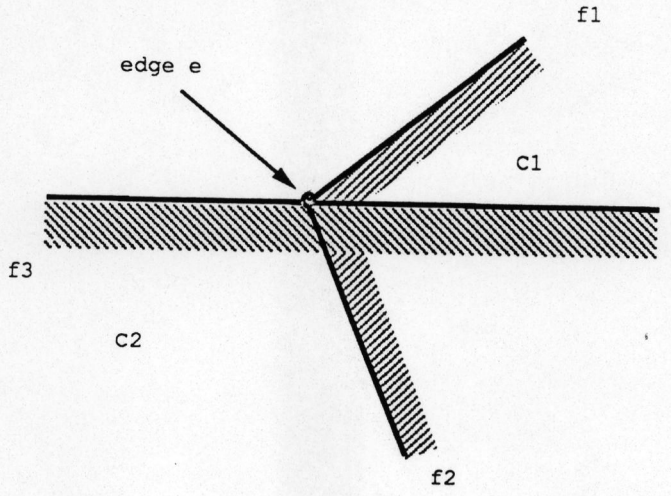

Figure 3: The example of removing indirect redundancy

In contrast to conventional boundary representation vertices are not shared among faces. For instance, for an n-sided pyramid, the top 'vertex' is actually not represented by one, but by n vertices, each is geometrically incident to three faces (the face it belongs to, and the two adjacent faces). These vertices are not considered coincident in 3 space, so redundancy is essentially avoided.

2.5 The robustness

The reason why most correctly designed geometric algorithms fail is that the decisions upon approximated data are arbitrary. When dependencies between such decisions are unrecognized and the decisions are made independently, dependent information may contain contradictions.

To eliminate inconsistencies in data generated by an algorithm, based on approximated data, we rely on the following lemma:

Lemma 1 *If an algorithm creates consistent and accurate results for correct input, on the grounds that all the computations are done with infinite precision, the same algorithm also produces consistent results, even with approximated data, if redundant data is eliminated before it affects any decision in the algorithm.*

The argument for proving the lemma is that, if we can avoid or remove redundant data

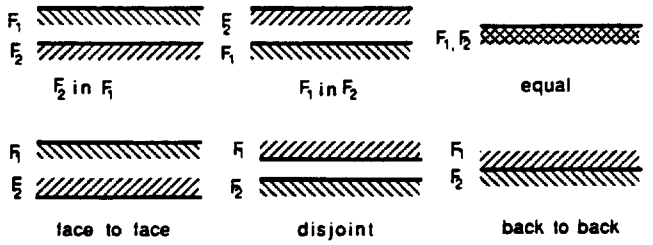

Figure 4: Degenerate cases: F_1^0 and F_2^0 are parallel

we avoid making dependent decisions. All decisions are therefore independent, and thus cannot be inconsistent.

To eliminate redundant data in the Boolean set operation algorithm we eliminate all coincident surfaces (by merging them). Curves incident on surfaces are limited to intersection curves between 2 surfaces, and vertices are intersections of three surfaces which is achieved by limiting the solids to 2 manifold topology, with each face having 1-manifold topology.

No coincident curves, and points will be generated in the representation (non-manifold objects are simply represented by a pseudo 2-manifold, disregarding coincidences).

3 Algorithms

The algorithm presented here will evaluate the elements of the boundary representation (vertices, edges and faces) from a solid object defined in half-space representation. First we compute the half space representation: We first check through all the faces, and merge them if they are coincident within the tolerance. The Boolean expression is brought to a disjunctive normal form (union of intersections of half spaces), as described above.

After this, we evaluate the boundary representations from it.

First, we evaluate the generalized convex bodies. A face is topological structure which is bounded by the edges. To compute all the faces, we just need to compute the edges that create the boundary of each face. The procedure "Faces", described below, computes the boundaries of all faces of a generalized convex body B.

```
Procedure Faces(B: GeneralizedConvexBody)
Begin;
  for each f1(j) in B do
```

```
    for each f2(k) in B where (k < j) do

        (* indices j,k are used to avoid directly redundant computation *)

    case spatial_relation(f1(j), f2(k) ) of
       intersecting:      e := PrimitiveEdge(B,f1(j), f2(k));
                          |
       f2_in_f1    :      delete f1(j)|
       f1_in_f2    :      delete f2(k)|
       equal       :      delete f1(j)|(* delete one of the two *)
       face_to_face:                  |
       disjoint    :      delete B; exit   |
       back_to_back:      delete B; exit   |

(* two half-spaces are equal or back to back if, they point
          to the same surface, after identical surfaces were merged
          during the previous evaluation of the half-space representation.
        *)
       end;
    od;
  od;
end Face;
```

We check the relationship of two half-spaces, which is either intersecting, parallel, or coincident (with equal or opposite orientation). Based on the condition that the body B is a convex body, we can delete superfluous faces (if they are completely outside B) or find out that B is actually an empty set. Only if the two half-spaces intersect each other, an intersection curve of the two half-spaces is created by the procedure "PrimitiveEdge".

After finding out the intersection curve e of the two half-spaces, we need to prune it by intersecting this curve with other half-spaces in this body. Either e intersects with a half-space so that we can compute an end point of the curve e, or e is parallel to or on the half-space, in which case a further evaluation of their spatial relation must be made (see figure 5). The relationship between the curve and the half-space will be provided by the procedure "curveSurfaceRelation(f1, f2, S, angle)". Faces f1 and f2 are intersecting in e, and

S is the third half-space.

```
Procedure PrimitiveEdge(B: ConvexBody; f1, f2: Face) :Edge;
Begin;
  e := intersectionCurve( f1, f2 );
  add e to f1(j); add e to f2(k)
  for each (* face *) S in B where ( S != f1 & S != f2 ) do
    case curveSurfaceRelation(f1,f2,S, angle) of
       intersecting: (* vertices are computed later by pair-wise
                        intersection of the edges within each face *)
       |
       on:
          case angle of
             2,3,4   : delete B; exit      |
             6,7,8   : delete S            |
             1       : delete e; delete f1 |
             5       : delete e; delete f2
          end;
       |
       outside:
          delete e
          case angle of
             2,3,4   : delete B; exit |
             8,1     : delete f1      |
             5,6     : delete f2      |
          else end;
       else end;
   od;
  return e;
end PrimitiveEdge;
```

Based on different angle position cases, shown in figure 5, we will remove all the redundant and superfluous half-spaces, along with their edges, as explained in the previous section.

Next, after computing the edges and faces of the GCBs we need to evaluate the union operations between them. We first compute the intersection edges between the faces of 2

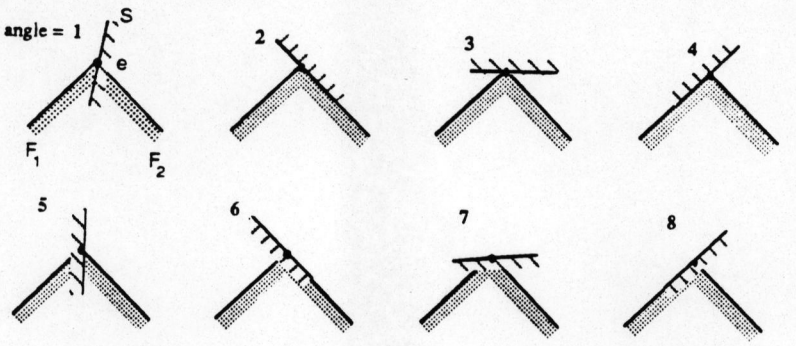

Figure 5: Classifying angle of S with respect to F1 & F2 (section perpendicular to the e shown)

GCBs and then compute their relations with the boundaries of both solids. The primitive edges need to be related only to the boundaries of the other solid. This operation will prune the edges and eliminate redundant ones. Faces whose edges have been completely eliminated will be deleted as well.

The result of the union of two CGBs is again a 2-manifold object which, however, is no longer necessarily convex. We take the result of the union and compute another union between it and the next CGB, and so on, until all CGBs are united.

The degenerate cases that can occur in this phase of the union operation are: An edge can be incident with one face, or, an edge can be coincident with another edge. No more than two edges can be coincident, since both input arguments always have 2-manifold topology.

The following two examples show how such degenerate cases are handled. (there are only very few possibilities that can occur). Figure 6 shows some typical cases in the pruning stage. In (a) the primitive edge e is the intersection of the faces f1 and f2 from the same convex body. Face f3 is a face from the other GCB. We solve this case by eliminating the edge e, and creating two edges f and g. Edge f is the intersection of face f1 and face f3, and edge g is the intersection of face f2 and f3. Note that this example would actually lead to non-manifold topology. We create a "pseudo" 2-manifold topology and regard the two edges as non coincident, such that they can be handled independently, later.

In (b) edge e is the intersection of face f1 and face f2 in one solid, and edge f is the intersection of face f3 and face f4 in the other solid. We solve this union by eliminating both edges, f and e and create a new edge g which is the intersection of f1 vs. f4.

After the edges have been computed without redundancy, we compute the vertices on each face by pair-wise intersection of the edges. The topological decisions made in this phase are almost identical to the decisions made for computing the edges, but one dimension lower

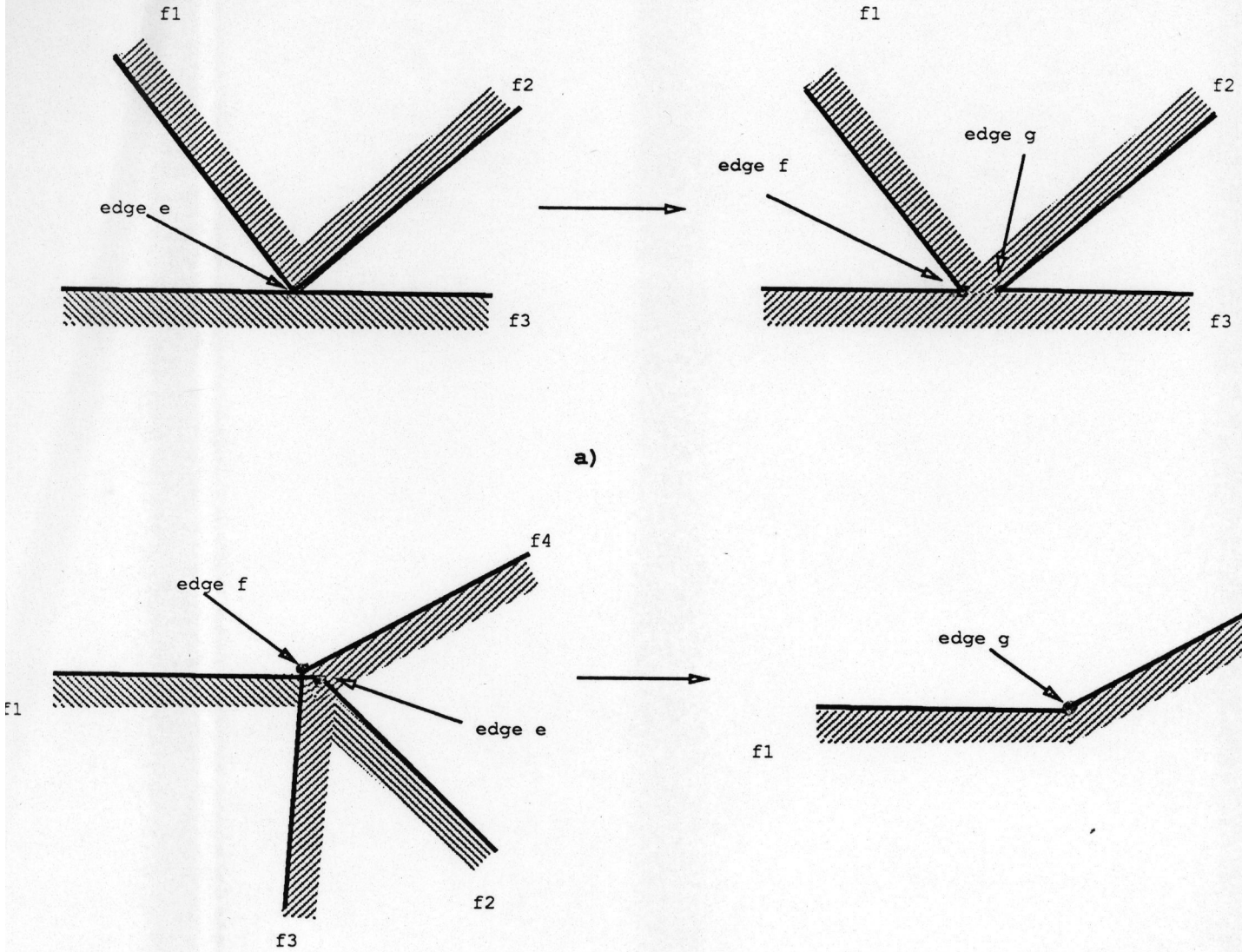

Figure 6: Two typical case happen in the pruning stage (section perpendicular to e shown)

(in a 2-dimensional hyperspace). The operations can be described like the ones for edges, above, by replacing "edges" with "vertices", and "faces" with "edges" in the text and the illustrations.

After computing the vertices, we can simply travel along the edges, and link them into rings. We consider only the case of 1-manifold face topology, therefore, each vertex will have exactly two oriented edges connected to it.

4 Conclusion

The algorithm for Boolean set operations presented in this paper computes the boundary representation of regularized Boolean expressions over half-spaces, represented by planar and natural quadric surfaces. All special (degenerate) cases are handled properly and robustly despite the fact that floating point arithmetic is used for which incidence decisions have to be made with some tolerance).

Robustness is achieved, here, by elimination of redundancy in the data representation which is possible for objects resulting in a 2-manifold topology.

The approach taken here has the advantage over previous approaches, that no explicit reasoning by the algorithm is necessary, and no ambiguities, which need to be resolved, can occur. This makes the approach simpler and easier to understand, and more efficient at the same time.

5 Acknowledgments

This work has been supported, in part, by NSF grants DDM-89 10229 and ASC-89 20219, and a grant from the Hewlett Packard Laboratories. All opinions, findings, conclusions, or recommendations expressed in this document are those of the authors and do not necessarily reflect the view of the sponsoring agencies.

References

[1] BRUDERLIN, B. Detecting ambiguities: An optimistic approach to robustness problems in computational geometry. Tech. Rep. UUCS 90-003 (submitted), Computer Science Department, University of Utah, April 1990.

[2] BRUDERLIN, B. Robust regularized set operations on polyhedra. In *Proc. of Hawaii International Conference on System Science* (January 1991).

[3] FANG, S. ZHU, X. AND BRUDERLIN, B. *Robustness In Solid Modeling - A Tolerance Based, Intuitionictic Approach* Tech. Rep. UUCS 92-030 (submitted), Computer Science Department, University of Utah, August,1992.

[4] CHOU, S. C. *Mechanical Geometry Theorem Proving.* D. Reidel Publ., Doordrecht, Holland, 1988.

[5] EDELSBRUNNER, H., AND MUCKE, E. Simulation of simlicity: A technique to cope with degenerate cases in geometric algorithms. In *Proc. of 4th ACM Symposium on Comp. Geometry* (June 1988), pp. 118–133.

[6] FANG, S. Robustness in geometric modeling – an intuitionistic and toleranced-based approach. Ph.D dissertation, University of Utah, Computer Science Department, 1992.

[7] FANG, S., AND BRUDERLIN, B. Robustness in geometric modeling – tolerance based methods. In *Computational Geometry – Methods, Algorithms and Applications, International Workshop on Computational Geometry CG'91* (March 1991), Springer Lecture Notes in Computer Science 553, Bern, Switzerland.

[8] FANG, S., AND BRUDERLIN, B. Robust geometric modeling with implicit surfaces. In *Proc. of International Conference on Manufacturing Automation, Hong Kong* (August 1992).

[9] GOLDMAN, R. N., AND MILLER, J. R. Combining algebraic rigor with geometric robustness for the detection and calculation of conic sections in the intersection of two natural quadric surfaces. In *Proc. of the ACM/SIGGRAPH Symposium on Solid Modeling Foundations and CAD/CAM Applications* (June 1991), Austin Texas.

[10] GREENE, D., AND YAO, F. Finite resolution computational geometry. In *Proc. 27th IEEE Symp. Fundations of Computer Science* (1986), pp. 143–152.

[11] GUIBAS, L., SALESIN, D., AND STOLFI, J. Epsilon geometry: Building robust algorithms from imprecise computations. In *Proc. of 5th ACM Symposium on Computational Geometry* (1989).

[12] HOFFMANN, C. M. *Geometric and Solid Modeling: An Introduction.* Morgan Kaufmann Publishers, 1989, ch. 4.

[13] HOFFMANN, C. M. The problems of accuracy and robustness in geometric computation. *IEEE Computer 22*, 3 (March 1989), 31–41.

[14] HOFFMANN, C. M., HOPCROFT, J. E., AND KARASICK, M. S. Towards implementating robust geometric computations. In *Proc. of 4th ACM Symposium on Computational Geometry* (June 1988), pp. 106–117.

[15] HOFFMANN, C. M., HOPCROFT, J. E., AND KARASICK, M. S. Robust set operations on polyhedral solids. *IEEE Computer Graphics and Application 9* (November 1989).

[16] KAPUR, D. Using grobner bases to reason about geometry. *J. Symbolic Comp. 2* (1986), 399–408.

[17] KARASICK, M. On the representation and manipulations of rigid solids. Ph.D thesis, McGill University, 1989.

[18] KUTZLER, B. Algebraic approaches to automated geometry proving. Ph.D Diss., Report 88-74.0, Research Institute for Symbolic Comp., Kepler University, Linz, Austria, 1988.

[19] LEVIN, J. A parametric algorithm for drawing pictures of solid objects composed of quadric surfaces. *Communications of ACM 19*, 10 (October 1976), 555–563.

[20] MILENKOVIC, V. Verifiable implementations of geometric algorithm using finite precision arithmetic. *Artificial Intelligence 37* (1988), 377–401.

[21] MILENKOVIC, V. Verifiable implementations of geometric algorithm using finite precision arithmetic. Ph.D thesis, Carnegie Mellon University, 1988.

[22] MILENKOVIC, V. Calculating approximate curve arrangement using rounded arithmetic. In *ACM Annual Symposium on Computational Geometry* (1989), pp. 197–207.

[23] MILENKOVIC, V., AND NACKMAN, L. R. Finding compact coordinate representations for polygons and polyhedra. In *ACM Annual Symposium on Computational Geometry* (1990), pp. 244–252.

[24] MUDUR, S. P., AND KOPARKAR, P. A. Interval methods for processing geometric objects. *IEEE Computer Graphics and Application 4*, 2 (February 1984), 7 - 17.

[25] OTTMANN, T., THIEMT, G., AND ULLRICH, C. Numerical stability of geometric algorithms. In *ACM Annual Symposium on Computational Geometry* (June 1987), pp. 119–125.

[26] REQUICHA, A. A. G. Representation for rigid solids: Theory, methods and systems. *Computing Surveys 12*, 4 (December 1980).

[27] SALESIN, D. Epsilon geometry: Building robust algorithms from imprecise computations. Ph.D thesis, Stanford University, 1991.

[28] SALESIN, D., STOLFI, J., AND GUIBAS, L. Epsilon geometry: Building robust algorithms from imprecise calculations. In *ACM Annual Symposium on Computational Geometry* (1989), pp. 208–217.

[29] SEGAL, M. Using tolerances to guarantee valid polyhedral modeling results. *Computer Graphics 24*, 4 (1990), 105–114.

[30] STEWART, A. J. Robust point location in approximate polygons. In *1991 Canadian Conference on Computational Geometry* (August 1991), pp. 179–182.

[31] STEWART, A. J. The theory and practice of robust geometric computation; or, how to build robust solid modelers. Ph.D Thesis 91-1229, Department of Computer Science, Cornell University, 1991.

[32] SUGIHARA, K., AND IRI, M. Geometric algorithms in finite precision arithmetic. Res. Mem. 88-14, Math. Eng. and Information Physicas, University of Tokyo, 1988.

[33] SUGIHARA, K., AND IRI, M. A solid modeling system free from topological inconsistency. *Journal of Information Processing 12*, 4 (1989), 380–393.

[34] TROELSTRA, A. S. *Constructivism in Mathematics : An Introduction.* Elsevier Science Pub. Co., 1988.

[35] YAP, C. K. A geometric consistency theorem for a symbolic perturbation theorem. In *Proc. of 4th ACM Symposium on Comp. Geometry* (June 1988), pp. 134–142.