NOTICE WARNING CONCERNING COPYRIGHT RESTRICTIONS:
The copyright law of the United States (title 17, U.S. Code) governs the making of photocopies or other reproductions of copyrighted material. Any copying of this document without permission of its author may be prohibited by law.

Non-Regularized Boolean Set Operations on Non-Manifold B-Rep Objects

by

E. L. Gursoz, Y. Choi, F.B.Prinz

24-18-90

Non-Regularized Boolean Set Operations on Non-Manifold B-Rep Objects

[1]

E, Levent Gursoz, Young Choi, and **Friedrich B. Prinz**

19 June 1990

Engineering Design Research Center
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

For Boolean operations on geometric models, we have developed an intersection algorithm for non-manifold boundary models with vertices, linear edges, planar faces, and volumetric regions. The algorithm operates by intersecting entities in an ordered manner, from vertex to edge, then to face elements, in contrast to intersecting pairs of faces as in two-manifold boundaries. Singular intersections are systematically handled by determining if an entity in one object is within a tolerance region of the entity in the other object. Two-manifold objects, i.e. solids, and non-manifold objects represented with the vertex-based data structure, which is designed for non-manifold representation, are handled by the same intersection algorithm. An implementation of the proposed algorithm and the experimental results are also discussed.

[1] ***This*** **research was supported in part by the Engineering Design Research Center, an NSF Engineering Research Center at Carnegie Mellon University**

UNIVERSITY LIBRARIES
CARNEGIE MELLON UNIVERSITY
PITTSBURGH, PA 15213-3890

1 Introduction

Boolean set operations on solid geometric models are convenient ways of modeling complex shapes. Simple primitives such as half-plane, block, cone, sphere, and torus are used to build desired shapes, with proper sequence of Boolean set operators. To apply any Boolean set operator for a pair of objects, it first is necessary to compute the intersection between two objects. Usually these intersection algorithms depend on the method used in representing the objects.

A number of intersection algorithms and implementations for polyhedral objects have been reported so far [1, 4, 7, 9, 10, 13, 18, 20, 21]. However, most of the algorithms are designed to be used for the regularized set operations with manifold objects [15]. Furthermore, very little has been published on the robustness issue of the intersection algorithm. Laidlaw et al [8] deal with the robustness issue, but their algorithm can handle objects with only convert polygon boundaries. Segal and Sequin [12] use variable tolerance to enhance the robustness of their algorithm. Hoffmann et al [6] introduce an incidence test for the robustness of their algorithm, but their algorithm is still limited to manifold objects, although they used a data structure that can represent non-manifold objects.

Recently, the advantages of a non-manifold representation have been recognized and several representation schemes have been developed [2, 5,11,16,17]. With the increasing popularity of non-manifold modeling, it is necessary to provide efficient tools for modeling and reasoning activity, e.g., calculating cross-sections of an object and determining mating surfaces between two solid objects. Boolean set operations, together with non-manifold geometric models, can serve as one such tool. In the non-manifold domain, the idea of regularized set operations is no longer required because different dimensional geometric entities can be mixed in one object. Instead, set theoretic Boolean operation must be considered for non-manifold objects. The set theoretic Boolean operation can allow open sets as a result. The set theoretic Boolean operation is closed in the non-manifold domain, in contrast to the non-closure in manifold domain. The set theoretic Boolean operation in non-manifold domain has been discussed [2,11], however the details of an implementation have never been published.

In this paper, we discuss an intersection algorithm for non-manifold objects. After intersection computation (keeping track of all the new topological relations), information gathered during the intersection process is used for set theoretic Boolean operations. The current algorithm and

implementation is limited to objects composed of points, straight lines and planar facets. Figure 1 shows an example of non-manifold objects that is built from simple geometric entities such as spheres, cones, straight line segments, and polygons.

The proposed algorithm computes the intersection at various levels of dimensionality instead of performing intersections between a pair of faces. With this algorithm, lower dimensional entities are investigated for intersections prior to the higher dimensional entities. This algorithm addresses all the special cases, such as vertex-vertex, vertex-edge, and edge-edge coincidence, without special treatment. Furthermore, this algorithm computes the intersection between non-manifold boundaries exactly the same way as manifold boundaries. We discuss our experience of the robustness of the algorithm with the current implementation at **the** end.

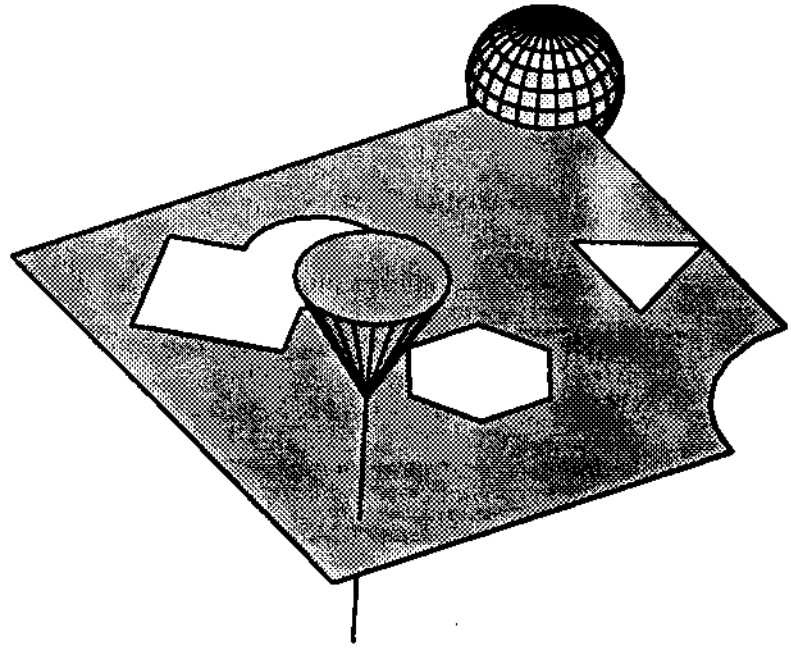

Figure 1: A Non-manifold Object with Linear Geometry

2 Representation of Objects

We chose a vertex-based data structure to represent non-manifold boundaries. A detailed description of the topological elements and the data structure can be found in [2,5]. In this paper, we discuss briefly only the important concepts and the point set elements that are relevant to Boolean set operations.

Non-manifold modeling in R^3 can be considered as exhaustively decomposing the R^3 into disjoint sets of elements of zero, one, two, and three dimensional point sets (i.e., vertices, edges, faces, and regions respectively). With these premises, the elements do not have any intersection in R^3_y and the union of all the elements always makes up R^3. Although additional topological elements have been introduced to capture the adjacency relationship between the point set elements explicitly [2, 5], the intersection can be viewed as recategorization of R^3 into these fundamental elements.

- A **vertex** is a zero dimensional point element defined by a position vector in R^3.
- An edge is a connected one dimensional element defined by its geometry and end points. In the linear modeling domain, an edge is defined by two end vertices only. An edge does not include its bounding vertices.
- A **face** is a connected two dimensional element defined by its geometry and bounding vertices and edges. In linear geometry domain, bounding vertices and edges on the same plane define a face. A face does not include its bounding vertices and edges. A face can be either convex or concave and have multiple loops. Figure 2 is an example of a face with concavity and four loops.
- A **region** is a connected three dimensional element defined by bounding vertices, edges, and faces. A region does not include its boundary elements and there is always one, and only one, region whose volume is infinite.

A non-manifold object can be any combination of the above point set elements. Therefore, set theoretic Boolean operation in non-manifold domain must be able to handle a pair of objects with any combination of the above point set elements.

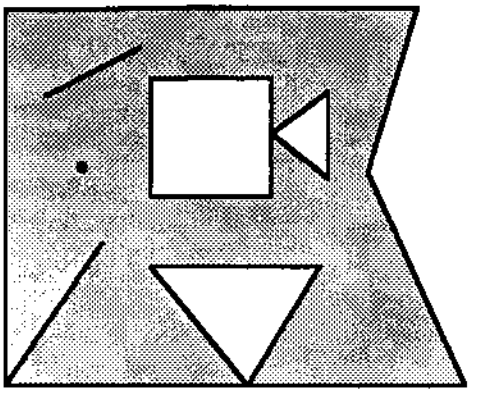

Figure 2: A Face with Multiple Loops

3 Set Theoretic Boolean Operations

Regularized set (r-set) operators in geometric modeling were originally introduced and formalized by Tilove and Requicha [14, 15]. Figure 3 shows an example of the r-set intersection and difference operations compared to the set theoretic Boolean operations with two polygons. The result of the set theoretic intersection has a dangling edge which violates the regularity of polygon. The result of the set theoretic difference has an open boundary which again violates the regularity of polygon. The regularized operations achieve the regularity of polygons by successively applying *interior* and *closure* operations to the results of set theoretic Boolean operations.

In our representation scheme, the collection of all the point sets in a model is mutually exclusive and collectively exhaustive in R^3. Since non-homogeneity is allowed in the representation, the set theoretic Boolean operations can be performed on a pair of objects. Even though the Boolean operators work on point sets, there are minimum units for the manipulation of the point set in the context of geometric modeling. In our representation, these units arc vertex, edge, face and region, as previously described.

Figure 4 is an example of set theoretic Boolean operations. Assuming that there are two intersecting objects, it is necessary first to subdivide the two objects into non-overlapping, disjoint point set elements. This procedure is accomplished by the intersection computation

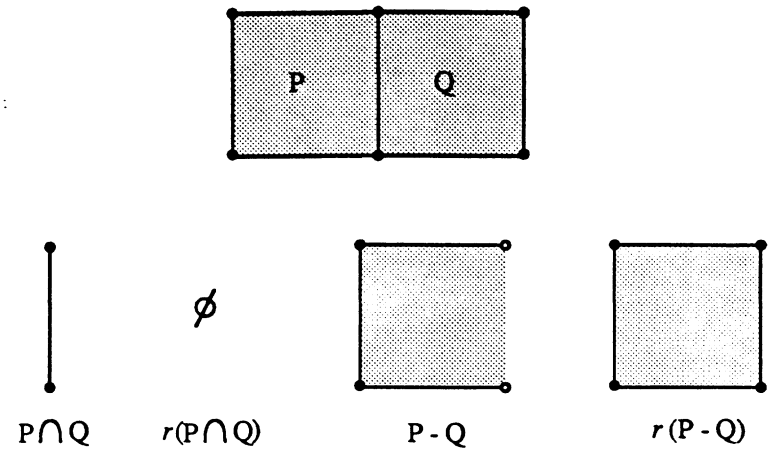

Figure 3: Set Theoretic and Regularized Intersection and Difference

discussed in the next section. A pair of objects is topologically merged in the intersection computation process. During the intersection computation, the element subdivision information is stored, and it is later used to select the proper point set elements for the result of Boolean set operations. In other words, when a certain point set element is subdivided into smaller, but equidimensional, elements and lower dimensional elements, all the information regarding this subdivision is stored.

Let's take the example of a square *A* as shown in Figure 4. By the intersection computation between objects *A* and *B*, a point set of face in object *A* (boundary is not included) is subdivided into two smaller faces, two edges and one vertex in the middle. Top and right side edges of object *A* are also subdivided into two shorter edges and one vertex each. Even though this is a two dimensional example, a three dimensional point set element *region* is treated in the same manner. Generally, a region can be subdivided into vertices, edges, faces and smaller regions. With this approach, Boolean set operation will be just a selection process among subdivided elements after merging two objects. As shown in the example, an open set also is allowed as a result. *Closure*, *purge* or *regularization* operation can be also used as a post-process if it is necessary [2, 11]. Following the merging and selection process for Boolean set operation, the unnecessary portion of the merged model may be deleted from the representation.

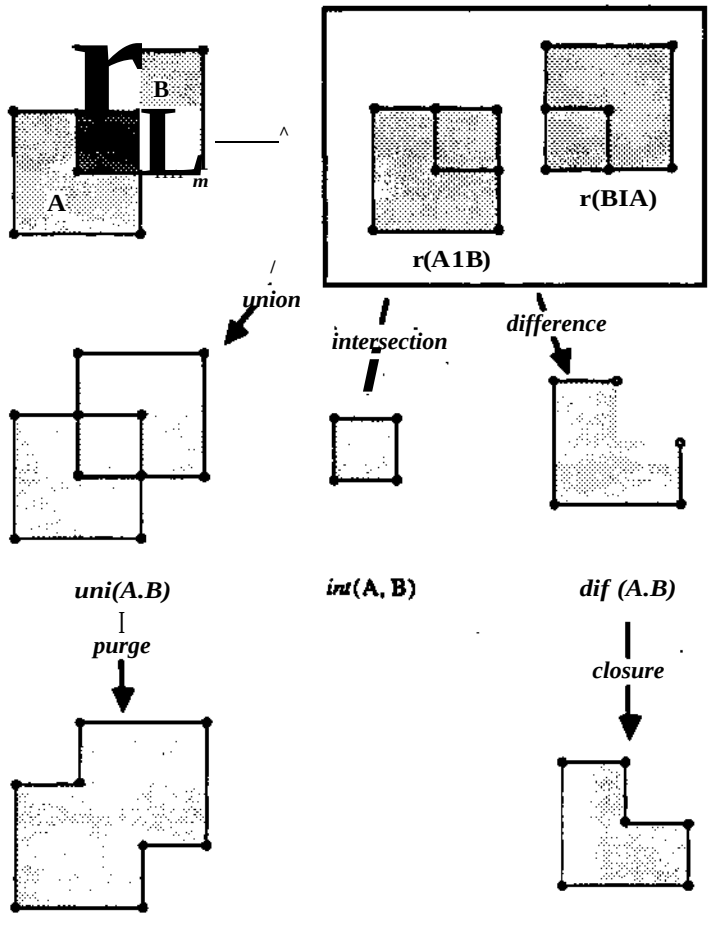

Figure 4: Set Theoretic Boolean Operations

4 Intersection Computation

There are 10 types of possible intersection pairs of point set elements because there are four different types of point set elements, namely vertex, edge, face, and region. However, intersection computation for the pairs associated with region can be accomplished by topological traversal around the intersecting boundary surface of the region. This is possible because in ihe B-rep scheme, a region is defined implicitly by the bounding surfaces. The region subdivision algorithm will be discussed in the next section. In this section, therefore, only 6 types of intersecting pairs described below are considered for actual intersection computation.

The basic idea of the proposed bottom-up intersection algorithm is to investigate intersections

between lower dimensional point set elements before proceeding with higher dimensional ones. Hence, the intersections are performed in the following order

1. Vertex-vertex intersection
2. Vertex-edge intersection
3. Edge-edge intersection and Vertex-face intersection
4. Edge-face intersection
5. Face-face intersection

All necessary topological modification is carried out completely during each intersection operation. For example, if there is an edge-edge intersection without coincident boundaries (vertices), then a vertex is created at the intersecting point and two edges are split with the new vertex. The result is four split edges, which are topologically connected through the new vertex. Face-face intersection is handled later with the lower dimensional intersection information. Whenever lower dimensional intersection is associated with a face, the information is stored for the computation of face-face intersections.

The procedure of intersection computation can be divided into two major parts. The first is intersection computation from the vertex-vertex pairs up to the edge-face pairs. The second is face-face intersection computation with the information obtained during the first part of the procedure. Overlapping faces, i.e., coplanar faces intersecting each other, are managed during the last step.

4.1 Vertex-Vertex Intersection

Vertex intersections, i.e., vertex coincidence, between two objects are investigated first. For each pair of vertices between two objects, if a distance between two vertices is less than the specified tolerance value, then those two vertices are topologically merged into one vertex. By merging these two vertices into one vertex, the position of the original vertices may be changed. Therefore, successive vertex merge operations with the same vertex may cause an undesirable effect called vertex drift. This effect can be reduced by using techniques such as taking the middle position between two vertices. Furthermore, the vertex merging may cause distortion of faces that are adjacent to those two vertices, because the position of at least one vertex of the two

must be moved

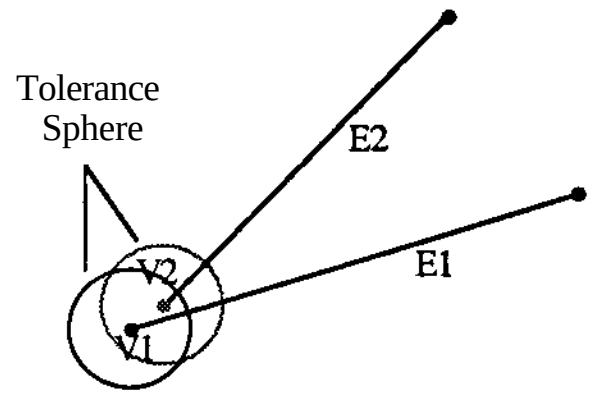

Figure 5: Intersection between Vertices

After merging, these coincident vertices are eliminated from each list of vertices so that these vertices are never considered again in vertex-vertex, vertex-edge and vertex-face intersection computation. The elemination means that intersection between lower dimensional elements is given higher precedence than the one between higher dimensional elements. In Figure 5, vertices *VI* and *V2* are topologically merged even if the vertex *V2* is closer to the edge *El* than the vertex *VI.* Intersection between the vertex *V2* and the edge *El* will not be considered later because of the elimination procedure described above. This action prevents the occurence of very short edges whose length is smaller than the tolerance value.

4,2 Vertex-Edge Intersection

If the distance between a vertex and its projected position on an edge is smaller than the specified tolerance value [Figure 6], then the edge is split using that vertex. Even though there is only vertex *V* in the example of Figure 6, generally edges and faces may be adjacent to the venex before intersecting. These edges and faces will be topologically connected to the other object through the vertex during split edge procedure. When the edge is split, the vertex may be moved to the projected position on the original edge to maintain the co-linearity of two split edges. But whether the position of the vertex is moved or not, geometrical instability, such as twisted face.

may occur.

The vertex is again deleted from the intersection candidate vertex list. The deletion prevents the vertex from being considered for intersection with faces that are adjacent to the edge. In Figure 6, the vertex V and the edge E intersect because the vertex V is in the tolerance region of the edge E. Intersection between the vertex V and the face F will never be considered in the later vertex-face intersection checking. A new split edge must be included in the candidate edge list for further intersection computation.

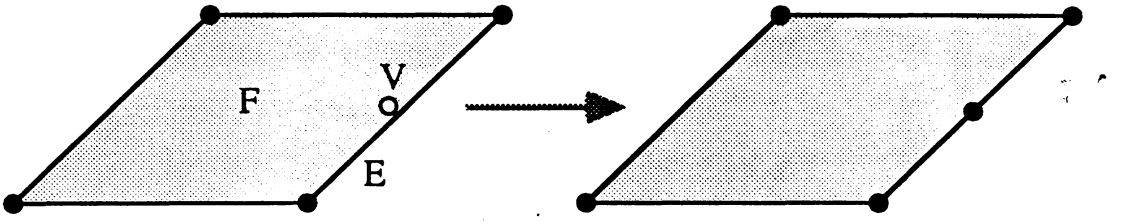

Figure 6: An Example of Vertex-Edge Intersection

4.3 Edge-Edge Intersection

After the vertex-edge intersection is completed, either edge-edge or vertex-face intersection is investigated. There is no significance in the order between edge-edge and vertex-face intersection computations. One type of intersection among these two does not have any propagating effect on the other type.

For edge-edge intersection, topological connectivity between edges is checked before going into the geometric intersection checking. If two edges are connected through a vertex, then edge-edge intersection computation is skipped. If two edges are connected at both ends, then two edges are topologically merged into a single edge. This is because all the edges are assumed to be linear. In the non-linear domain, however, edge-edge intersection must be computed regardless of the connectivity at the boundaries. These two cases of connected edges are results of vertex-vertex, vertex-edge or previous edge-edge intersection computation.

If a pair of edges passes the topological connectivity test, i.e., if they are not connected, and

the shortest distance between two edges is smaller than the specified tolerance value, then a new vertex is created at the intersection point and two edges are split with the new vertex. **The** position of **the new vertex** can **be** located either at the middle of the two edges or at one of the edges. In either case, however, some geometrical distortion is unavoidable. When two edges are split with the new vertex, all four split edges are topologically connected through that vertex. Two new split edges are added to each of the intersection candidate edge lists for further intersection computation.

4.4 Vertex-Face Intersection

At this point, all the vertex-vertex and vertex-edge intersections have been investigated and all the intersecting vertices have been deleted from the candidate vertex lists. This implies that it is not expected to compute intersections between a face and a vertex that is inside of the tolerance region of the face boundaries. Therefore, the result of vertex-face intersection is that either the vertex and face intersect or they do not intersect, but not a border case. The situation in Figure 6 will never be considered as vertex-face intersection because it has been already taken care of by vertex-edge intersection computation.

If the distance between a vertex and its projected position on a face is smaller than the specified tolerance value, and if the projected position is inside of the face, then the vertex is considered to coincide with the face. No immediate topological modification is carried out in this case, but this intersection information, including the vertex and the face, is stored. This intersection information is later retrieved and used for splitting or modifying the face.

4.5 Edge-Face Intersection

Finally, the edge-face intersection is investigated. As in the case of edge-edge intersection, topological connectivity **between** the boundary of the edge and the boundary of the face is checked. If they are connected through the boundaries, then the edge-face intersection computation is skipped The computation is skipped because there is an intersection between those two boundaries already. If there is no connectivity between the boundaries, then the geometric intersection between the edge and the face is computed.

If two end vertices are on the same side of the plane that the face is lying on, there is no intersection. Otherwise, the intersection point between edge tangent vector and the plane is computed. If the intersection point is inside the face, a new vertex is created and the edge is split

with the new vertex. No immediate topological modification is carried out for the face, but the intersection information, including the new vertex and the face, is stored. This intersection information is later retrieved and used for splitting and modifying the face.

4.6 Face-Face Intersection

So far, all the intersections except the face-face intersection have been completely resolved. And all the necessary information for the intersection line for intersecting faces have been recorded. Therefore, the next step is forming intersection lines for intersecting faces and actual split face operation. Overlapping faces are handled after the split face procedure. The split face operation can be divided into three sub-operations:

1. Intersection lines for a intersecting face are formed with the intersection record that was stored during previous intersection operations at various levels;

2. The face is cut into multiple faces along intersection lines or is just modified as another topologically different face;

3. Overlapping faces must be detected and topologically merged into a single face.

Each sub-operation is discussed in the following sections in detail.

4.7 Intersection Line for Non-coplanar Faces

There are two different types of intersections between two faces. One is intersection between two coplanar faces and the other is intersection between two faces that are not coplanar. These two cases are handled separately because intersecting coplanar faces do not need to create new edges for split face operations. Intersection between coplanar faces will be discussed in the next section.

For each pair of faces which intersect each other and are not coplanar, intersection lines are formed with pre-stored intersection records. All the intersecting vertices are identified and aligned along the intersection line [Figure 7]. Then, each pair of adjacent intersecting vertices is visited in turn to classify the intersection.

If there is an edge between any two consecutive vertices, then the edge is flagged so that it is included in the boundary of new faces resulting from the intersecting face. A pair of consecutive

vertices, *V3* and V4, in Figure 7 falls into this category. The face *Fl* is not affected by this intersection, but the face *F2* has to be modified so that the new face(s) includes the edge between the vertices as **a part of its(their) boundary.**

If there is **no connecting edge between two** vertices **and the mid-point** between these two vertices is inside both of the intersecting faces, then two faces actually intersect between these two vertices. Therefore, a new edge should be created between those two vertices and flagged so that it can be used in splitting or modifying both faces later. Vertex pairs *VI* and *V2,* and *V7* and *V8* in Figure 7 fall into this category. Two new edges must be created between *VI* and V2, and *V7* and *V8.* These two edges will be used later in reforming the faces *Fl* and *F2.*

Remaining vertices on the intersection line must be checked to see that each vertex does not belong to the boundary of the one of the intersecting faces. If a vertex does not, it is flagged so that it is added to the boundary of the face that is not associated with the vertex. Vertices *V5* and *V6* fall into this category. These two vertices do not affect the face F/, but they must be added to the boundary of the face *F2* later.

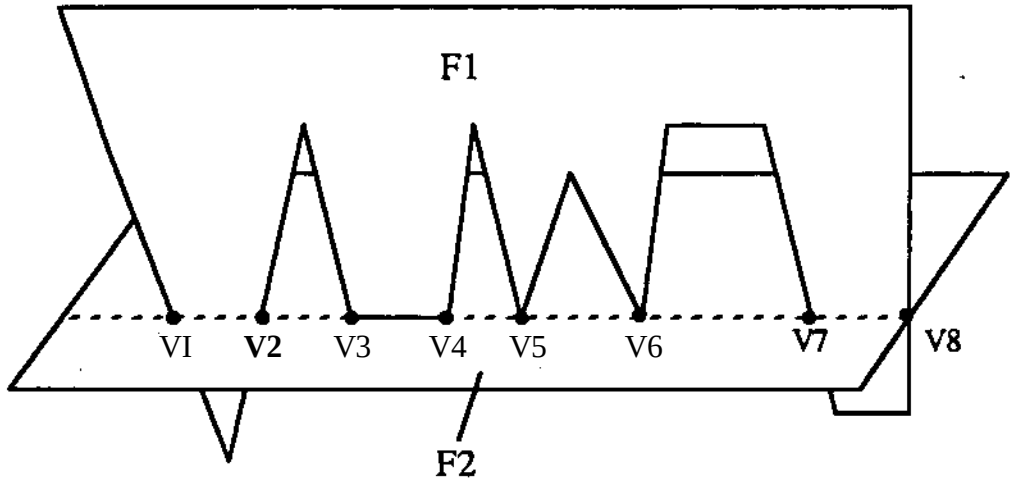

Figure 7: Intersection Line between Two Faces

4.8 Intersection Line for Coplanar Faces

If a pair of faces intersect each other and are coplanar, then the faces are checked to determine if a face geometrically contains edges or vertices from the boundary of the other face. If such edges or vertices exist, then they are flagged so that they are used as boundary elements for the face when the face is split. This process should be done for both faces. An example of overlapping faces is illustrated in Figure 8. In this example, faces *F1* and *F2* are overlapping in the shaded region. Edges *E1* and *E2* should be flagged to be used in splitting face *F1*, and edges *E3* and *E4* should be flagged to be used in splitting face *F2* later.

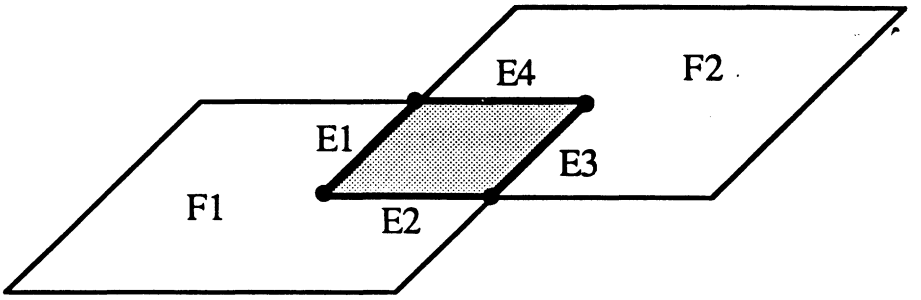

Figure 8: Overlapping Faces

4.9 Face Splitting

Before going into the details of the split face operation, the definition of face is revisited for clear explanation of the loop (oriented boundary) formation method. A face is defined by directed loops, where the outer loop has counter clockwise direction and the inner loop has a clockwise direction around normal of the face, following the right hand rule. A loop is defined by a directed cyclic list of connected edges and vertices. A face may or may not have inner loops, but it must have one outer loop and it should enclose an area greater than zero. All the edges in the loop should be adjacent to the enclosed area. In addition, an edge cannot lie inside a face if the edge is not a member of one of the loops of the face.

The first step in splitting a face is the formation of new loops from edges and vertices which

intersected with that face. The edges and vertices arc determined from steps of the algorithm explained in the preceding sections. A new edge is used in both directions in the loop formation because it lies inside the face. Therefore, both directions of new edges are added to a list of appropriate directed edges from the original loops of the face.

Now a list of unordered, directed edges is available for the formation of new loops. After picking up any directed edge in the list, the next loop direction should be decided at the front end of the directed edge. A strategy for correct loop formation using Edmonds technique [19] is illustrated at the top of Figure 9. In this figure, the face normal is coming out from the paper and all the candidate loop directions are drawn at the junction. In this case, direction £>/, which is closest from the left side of the incoming direction, is chosen. If any other direction is chosen, then the loop will violate the definition of the face. Once a loop is completed by closing itself, those directed edges are deleted from the list. This process continues until the list becomes empty. An intersecting vertex is a special case and it forms a loop by itself, as in example *A*.

At this point, all the legitimate loops are formed for new faces. However, these loops should be properly grouped to form a new set of faces. Adjacency information between loops and geometric information are used for this grouping operation. Outer loop and inner loop can be distinguished by the sign of their areas because their direction is opposite around the given face normal. An area of outer loop should be greater than zero and an area of inner loop should be equal to or less than zero.

The grouping operation starts with picking a loop with the smallest positive area. Then, candidate inner loops whose absolute values of the areas are less than the area of the outer loop are collected. Next, a geometric test is performed to check whether a candidate loop lies inside the picked'outer loop. A new face is formed from the outer loop and inner loops which passed the inclusion test. And these loops for the new face are deleted from the list of loops. This process is done exhaustively for all the loops until the list of loops becomes empty.

Some examples of loop and face formation are illustrated in Figure 9. In each example, solid lines are the original edges that come as boundaries of the original face and dotted lines are the new edges that must be used as a part of the boundaries for newly formed face. In example M, new edges and a vertex are introduced in a rectangular face as a result of the intersection. But the face remains as a single face with two inner loops. Example *B* is a case of simple face cut.

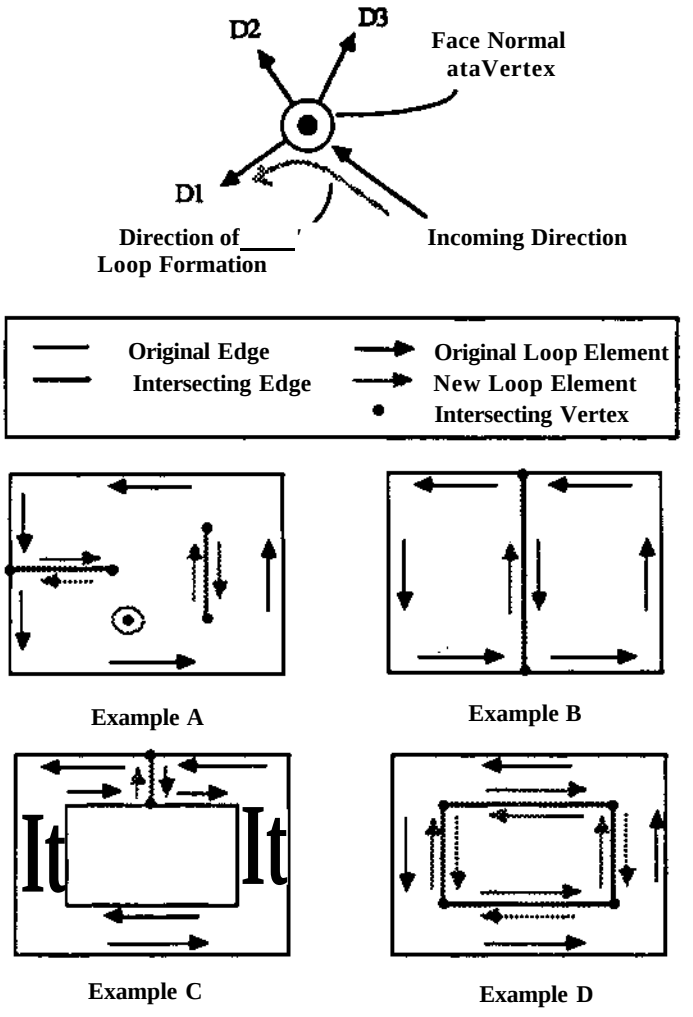

Figure 9: Formation of Loops for Intersecting Faces

The original rectangular face is split into two faces. In example C, an original face with a hole(inner loop) becomes a face with one loop after intersection. In example D, an original rectangular face is split into two faces. One is a face with a hole inside and the other is a smaller rectangular face. More arbitrary and complex face cut situations can be resolved with the same algorithm.

After all the faces are split properly, overlapping coplanar face pairs are detected by topological connectivity around the boundaries and are merged into a single face.

5 Region Subdivision

During **the intersection** procedure, subdivision information regarding vertices, edges, and faces is obtained and stored for the subsequent selection process of Boolean set operations. However, subdivision of regions (three dimensional point set elements) has not occurred yet. Intersecting regions may be subdivided into smaller regions, faces, edges, and vertices in the most general case. Figure 10 shows three examples of how the inside region of a cube can be subdivided. In the first example, a cube intersecting with another cube, the region inside a cube is subdivided into two smaller regions, three faces, three edges, and one vertex. If the second cube happens to be completely inside the first cube, then the inside region of the first cube will be subdivided into two smaller regions, six faces, twelve edges and eight vertices. In the second example, a cube intersecting with a rectangular face, the region is subdivided into one region, two edges, and one vertex. In the third example, a cube intersecting with an edge whose one end vertex is inside the region, the region is subdivided into one region, one edge, and one vertex. If both end vertices of the edge were outside the cube, the region would be subdivided into one region and one edge only.

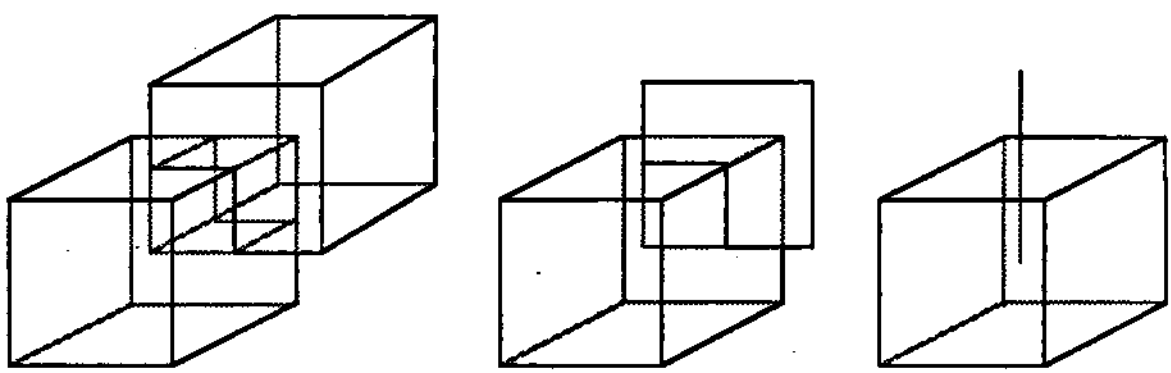

Figure 10: Several Types of Region Subdivision

For each region *R* in each operand object 0, the subdivision of region *R* is a process of collecting a corresponding point set 5 in the merged object A/. The object *M* already has subdivision information of faces, edges, and vertices as previously discussed. For each region /?. the subdivision process is performed in the following way:

1. Collect all the boundary point set elements of region *R* in object *O* and call it *B.*

Faces among these boundary elements have the orientation information for the region R.

2. Find corresponding point set elements of B in the merged object M and call it B_{shell}. The orientation information of B is inherited in B_{shell} in the process.

3. Find all the regions (R_{div}s) in merged object M which are partly or entirely bounded by B_{shell} and add these regions in S.

4. Collect boundary elements B_{div} for each region R_{div}. Again, faces have the orientation information for the R_{div}. If point set elements, that are not in B_{shell} and S, exist in B_{div}, add those in S.

5. Collect all faces F if there exist faces in B_{div} that are not in B_{shell} and only one side of the faces are used for B_{div}, and continue to next step. Otherwise stop here. The existence of F implies that there is a nested region (isolated shell) as a result of the region intersection.

6. Take opposite sides of the faces F that were used for B_{div} and call it B_{shell}, then go back to step 3.

The resulting set S contains all the point set elements that are subelements from a region R. The above algorithm can handle all the possible region intersection cases such as region-region, region-face, region-edge, and multiple nested region intersections (region intersection with no boundary shell intersection) without exception. After all regions are completely subdivided with the above algorithm, Boolean set operations simply select the proper point set elements from the object M according to the type of operation.

6 Tolerance

In the proposed intersection algorithm, a uniform floating point tolerance is used for all the boundary elements, i.e., for vertices, edges and faces. A dynamic tolerance, which changes its tolerance value depending on the scale of objects, may be used as an alternative. Because of the floating point tolerance, the effect of the tolerancing may depend on the position and the size of intersecting objects.

Tolerance regions for various point set elements are illustrated in Figure 11. A vertex has a tolerance region which is a sphere with radius *t*. An edge tolerance region is a cylinder with the same length as the edge and radius *t*. But the edge tolerance cylinder is truncated by a sphere with a radius *t* at each end as illustrated in Figure 11. This implies that a vertex claims the adjacent region as its tolerance prior to connected edges. A face tolerance region is basically a slab with thickness *It* and the same area as the face. However, the vertex tolerance region and edge tolerance region are subtracted from the slab along the boundaries. Therefore a face has a lower priority in claiming adjacent regions as its tolerance regions than vertices and edges. Appropriate prioritization of the intersections achieves the truncations in the tolerance regions.

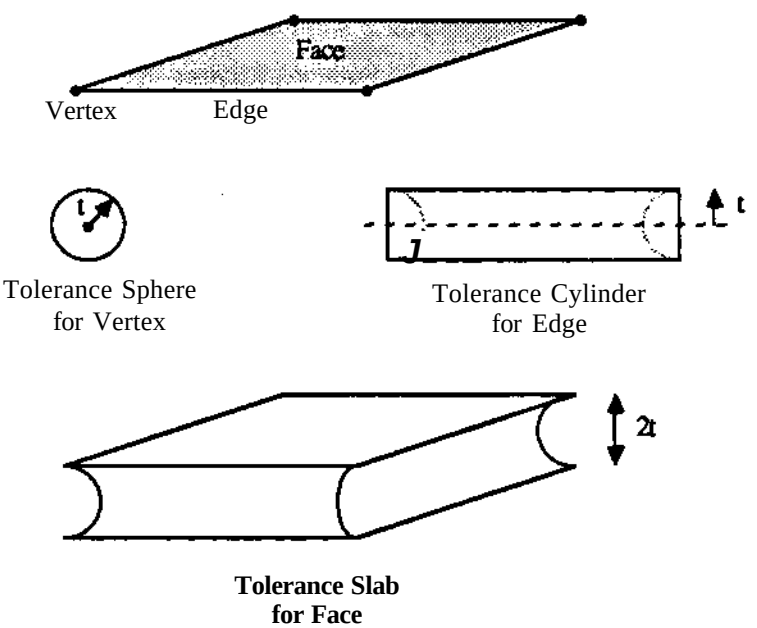

Figure 11: Tolerance Regions for Vertex, Edge and Face

With this tolerancing strategy, there will not be overlapping or missed regions in the vicinity of vertices, edges, and faces. In the proposed algorithm, consistent tolerancing with tolerance regions described above is automatically achieved by the *bottom-up sequence* of intersection computation with a tolerance value *t*.

7 Implementation and Experimental Results

The implementation of the proposed intersection algorithm and the Boolean set operations on non-manifold objects were carried out as a part of the NOODLES geometric modeling system project at the Engineering Design Research Center at Carnegie Mellon University. The program is written in the *C language and was tested on various hardware platforms, including Sun, Apollo, and DEC.

The robustness of the proposed algorithm is compared with previously published algorithms, although those algorithms were not designed for set theoretic Boolean operations on arbitrary non-manifold objects. Intersection between two unit cubes has been used as a measure to evaluate robustness of algorithms [6, 8]. One of these unit cubes is rotated by a certain angle around each principal axis successively before intersection.

There are two critical rotation angles when computing the intersection between these two cubes. The Boolean intersection between two cubes will be the same as the unit cube when the angle is smaller than a certain small value a. The Boolean intersection, when the angle is larger than a certain small value (3 but larger than a, will result in the formation of a polyhedra with 16 vertices. Between the angles a and P, the intersection may result in topologically incorrect objects or failure.

Laidlaw, Trumbore, and Hughes achieved the critical range (p - a) = 0.4 with their heuristics. Hoffmann, Hopcraft, and Karasick obtained better results of the critical range (p - a) = 10^{-10} with e = 0.6 with the incidence test. Segal and Sequin performed the same experiment and obtained a = 10^{-4} and p = 10^{-3}. However, all of the previous algorithms work only for solid objects, or polygons (for graphic rendering purpose). Furthermore, Laidlaw et al's algorithm was not capable of handling non-convex polygonal faces, and Hoffmann et al made an assumption that all the vertices are valence-three vertices.

With the current implementation, we obtained a = 10^{-4} and p = 10^{-1} with tolerance e = lOr^{6} on a Sun 3 workstation. Although this particular test result for cubes is not better than Hoffmann et al's and Segal and Sequin's, it is difficult to judge robustness based on one particular test for several reasons. First, heuristics that can be applied to the intersection computation between manifold objects is no longer valid in the non-manifold domain. Second, the chance of failure in computing geometry and topological adjacency at the intersection might be higher because the

non-manifold representation has more topological elements, such as *disks, zones,* etc., and is more complex than most manifold representations [2,5]. However, new heuristics for non-manifold **objects, variable tolerance, and dynamic tolerance** can be adopted in the proposed algorithm to further enhance the robustness.

Figure 12 shows set theoretic Boolean intersection results for two unit cubes with various relative positions. Figure 13 shows Boolean union, difference, and intersection operations between a sphere *(3D)* and a surface structure *(2D)* that does not have any enclosed volume. The surface structure was also constructed by Boolean union of 20 square faces in each x and y direction. Figure 14 shows an example of cross-section computation which can be used in layer-by-layer manufacturing processes such as streolithography or selective laser sintering [3]. The cross-sections were computed by **Boolean intersection between** L-shaped bracket and stacks of planar faces.

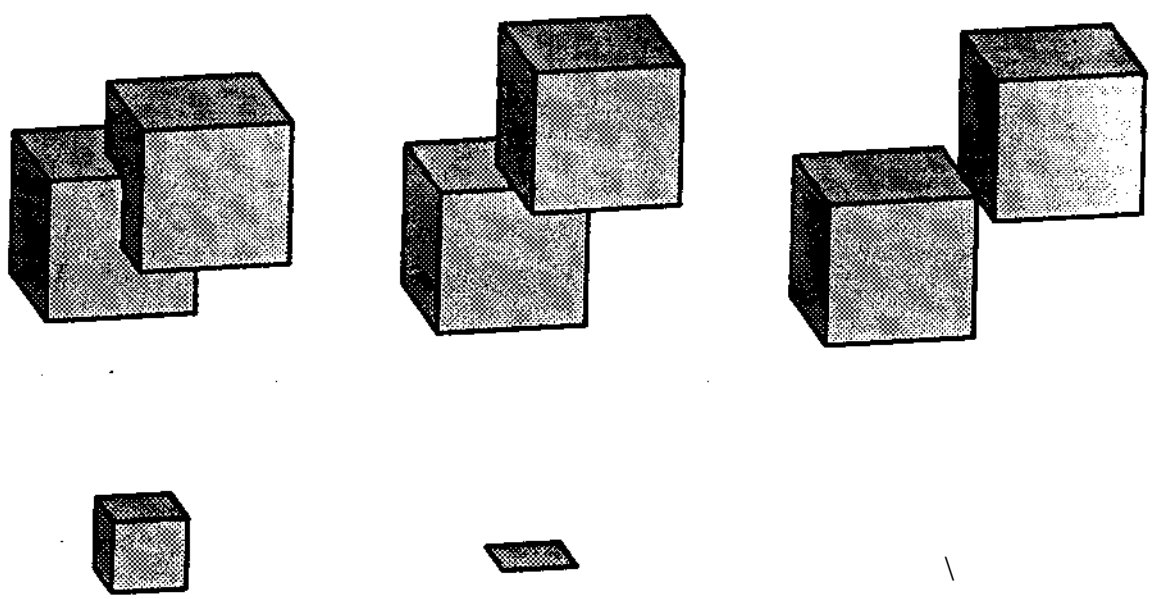

Figure 12: Boolean Intersection between Two Unit Cubes with Various Relative Positions

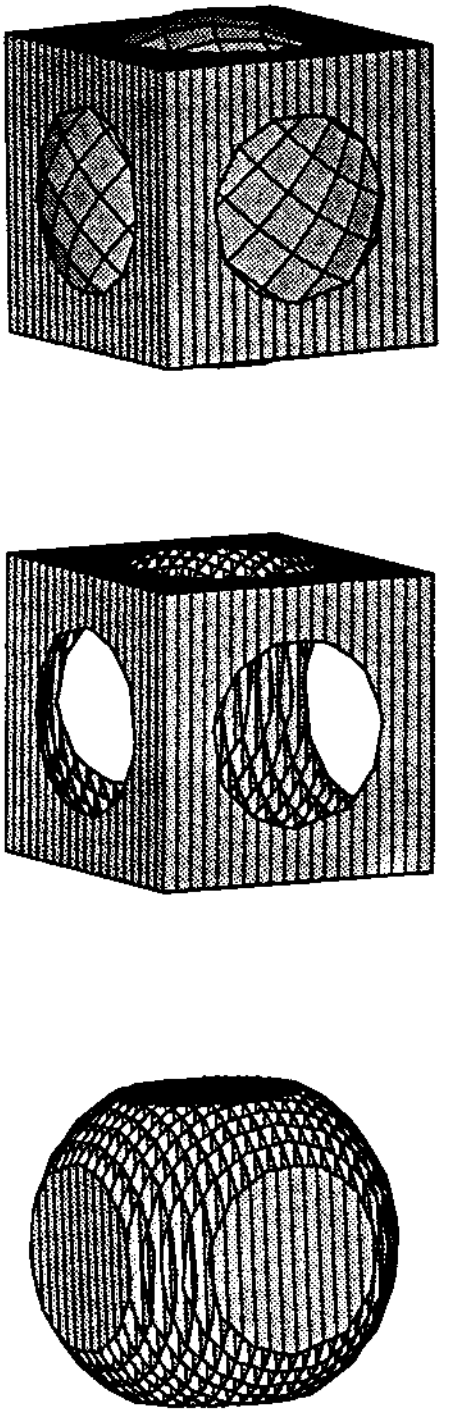

Figure 13: Boolean Operations between a Sphere and a Honeycomb-like Structure: Top object is a Boolean union of a sphere and a surface structure constructed by 20 square faces both in *x* and *y* directions. Results of Boolean difference and intersection are at the middle and the bottom, respectively.

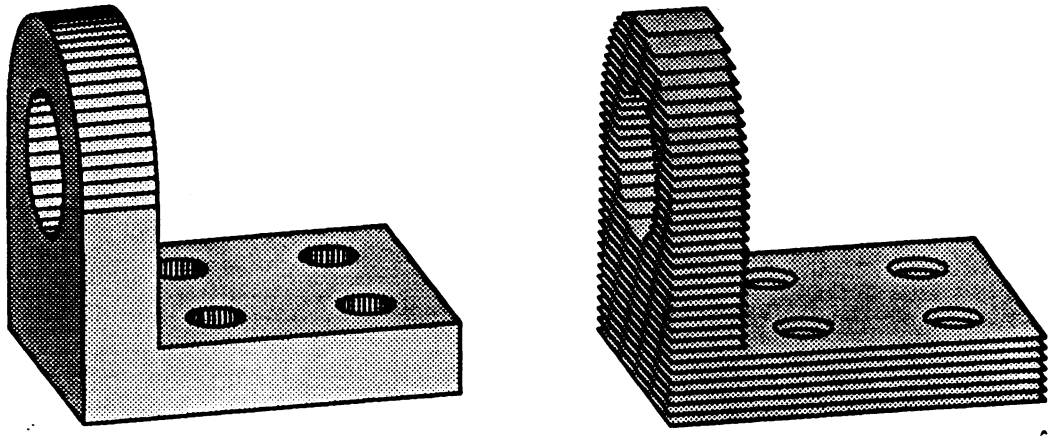

Figure 14: Cross-Section Computation by Boolean Intersection between a L-Shaped Bracket and Cross-Sectional Planes

8 Conclusion

As non-manifold geometric modeling receives more attention, it is necessary to develop ways to manipulate non-manifold objects. There are various ways to manipulate objects, such as Boolean set operations, sweeps, blending, and offsetting. This paper focused on the Boolean set operations. Since regularized Boolean operations on manifold objects are not sufficient in the non-manifold domain, it becomes necessary to handle objects with set theoretic Boolean operations. The set theoretic Boolean operations on non-manifold objects can be utilized in various geometric applications, such as geometric reasoning for manufacturing process planning and building design.

In this paper we propose an algorithm for the intersection computation and set theoretic Boolean operations on non-manifold objects with linear geometry. The algorithm operates through intersecting entities in an ordered manner, from vertex to edge, then to face elements, in contrast to intersecting pairs of faces as previously done for two-manifold boundaries. The prototype implementation showed similar robustness compared to previous manifold intersection algorithms which have significant assumptions and restrictions. Some graphical results from the

implementation were shown regarding the robustness of the algorithm. The authors believe that enhancements can be made by adding various tolerance techniques and heuristics.

References

[I] **L C Braid**
The Synthesis of Solids Bounded by Many Faces.
Communications of the ACM 18(4):209-216, April, 1975.

[2] Y. Choi.
Vertex-based Boundary Representation of Non-manifold Geometric Models.
PhD thesis, Department of Mechanical Engineering, Carnegie Mellon University, August, 1989.

[3] D. Colley.
Instant Prototypes.
***Mechanical Engineering* :68-70, July, 1988.**

[4] J. Flaquer, A. Carbajal, and M. A. Mendez.
Edge-edge Relationships in Geometric Modelling.
Computer-Aided Design 19(5):237-244, June, 1987.

[5] E. L. Gursoz, Y. Choi and F. B. Prinz.
Vertex-based Representation of Non-manifold Boundaries.
In M. Wozny, J. Turner and K. Preiss (editors), *Geometric Modeling for Product Engineering*, pages 107-130. North Holland, January, 1990.
Selected and Expanded Papers from the IFIP 5.2/NSF Working Conference on Geometric Modelling, Rensselaerville, U.S.A. 18-22 September, 1988.

[6] C. M. Hoffmann, J. E. Hopcraft and M. S. Karasick.
Robust Set Operations on Polyhedral Solids.
***IEEE Computer Graphics and Applications* :50-59, November, 1989.**

[7] J. Kripac.
Algorithm for Splitting Planar Faces.
Computer-Aided Design 19(6):293-298, July, 1987.

[8] D. H. Laidlaw, W. B. Trumbore and J. F. Hughes.
Constructive Solid Geometry for Polyhedral Objects.
***Computer Graphics (Proc. SIGGRAPH)* 20(4):161-170, August, 1986.**

[9] M. Mantyla.
Boolean Operations of 2-Manifolds through Vertex Neighborhood Classification.
***ACM Transactions on Graphics* 5(1): 1-29, January, 1986.**

[10] A. A. G. Requicha and H. B. Voelcker.
Boolean Operations in Solid Modeling: Boundary Evaluation and Merging Algorithms.
Proceedings of the IEEE 73(l):30-44, January, 1985.

[II] J. R. Rossignac and M. A. O'Connor.
SGC: A Dimension-Independent Model for Pointsets with Internal Structures and Incomplete Boundaries.
In M. Wozny, J. Turner and K. Preiss (editors), *Geometric Modeling for Product Engineering*, pages 145-180. North Holland, January, 1990.
Selected and Expanded Papers from the IFIP 5.2/NSF Working Conference on Geometric Modelling, Rensselaerville, U.S.A. 18-22 September, 1988.

[12] **M Segal and C H. Sequin.**
Partitioning Polyhedral Objects into Nonintersecting Parts.
***IEEE Computer Graphics and Applications* :53-67, January, 1988.**

[13] W. C. Thibault and B. R Naylor.
Set Operations on Polyhedra Using Binary Space Partitioning Trees.
***Computer Graphics (Proc. SIGGRAPH)* 21(4):153-162, July, 1987.**

[14] R. B. Tilove and A. A. G. Requicha.
Closure of Boolean Operations on Geometric Entities.
Computer-Aided Design 12(5):, September, 1980.

[15] R.B. Tilove.
Set Membership Classification: A Unified Approach to Geometric Intersection Problems.
IEEE Transactions on Computers c-29(10):874-883, October, 1980.

[16] K.J.Weiler.
Topological Structures for Geometric Modeling.
PhD thesis, Rensselaer Polytechnic Institute, 1986.

[17] K.J.Weiler.
The Radial Edge Structure: A Topological Representation for Non-Manifold Geometric Modeling.
In M. Wozny, H. McLaughlin, and J. Encarnacao (editors), *Geometric Modeling for CAD Applications,* pages 3-36^ North-Holland, 1988.
(IFIP WG5.2 Working Conference Rensselaerville, N.Y., 12-14 May 1986).

[18] M. A. Wesley, T. Lozano-Perez, L. I. Lieberman, M. A. Lavin, and D. D. Grossman.
A Geometric Modeling System for Automated Mechanical Assembly.
***IBM Journal of Research and Development* 24(l):64-74, January, 1980.**

[19] A. White.
Graphs, Groups, and Surfaces, Mathematical Studies 8.
North Holland, Amsterdam, 1973.

[20] F. Yamaguchi and T. Tokieda.
A Unified Algorithm for Boolean Shape Operations.
***IEEE Computer Graphics and Applications* :24-37, June, 1984.**

[21] F. Yamaguchi.
A Unified Approach to Interference Problems Using a Triangular Processor.
***Computer Graphics (Proc. SIGGRAPH)* 19(3):141-149, July, 1985.**