Incremental Boundary Evaluation For Nonmanifold Partially Bounded Solids

James R. Miller
Department of Electrical Engineering and Computer Science
University of Kansas
415 Snow Hall
Lawrence, KS 66045
email: miller@eecs.ukans.edu

Abstract

In an earlier paper we described an incremental boundary evaluation algorithm in which we were able to avoid the vast majority of explicit edge classifications by using an inference mechanism [13]. That algorithm only worked properly if the boundaries of the input and output solids were compact 2-manifolds. In this paper we lift those restrictions by describing modifications to that algorithm which allow input and output solids to be nonmanifold and/or only partially bounded. Both the original algorithm and the extended one described here allow solids bounded by portions of curved surfaces.

Keywords: Solid Modeling, Boundary Evaluation, Constructive Solid Geometry, Partially Bounded Solids.

1.0 Introduction

Solid Modeling has become an important tool in modern industrial design and manufacture. The ability to create and manipulate computer-based mathematical models of physically realizable solid objects saves industry time and money by allowing a variety of preliminary design options to be studied and simulations to be performed without constructing actual scale models. Many different representations for solid models have been described [15]. Although it is becoming increasingly difficult to place solid modeling systems and their representations into simple categories, it is generally acknowledged that two basic types of representation are by far the most prevalent in systems: Constructive Solid Geometry (CSG) and Boundary Representations (BReps).

A critical algorithm in any solid modeling system, whether it is internally based purely on BReps or some CSG-BRep combination, is the one which converts a CSG representation into the equivalent BRep. Basic boundary evaluation and merging algorithms have been known for some time [16]. In [13] we presented an algorithm for boundary evaluation based on inference of edge classifications, and we showed how it works for traditional bounded manifold solids. Here we extend this algorithm to admit unbounded solids with boundaries which are not necessarily manifold.

By "nonmanifold" we mean solids with a well-defined interior and exterior but with boundaries which are not 2-manifolds. Similarly when we say "partially bounded", we mean well-defined solids, but ones which may have infinite volume and/or infinite surface area. In both nonmanifold and partially bounded cases we are dealing with r-sets [15]. That is, we do not imply by either nonmanifold or partially bounded that we are considering solids with extra (e.g., "dangling") or missing vertices, edges, or faces.

We are interested in nonmanifold solids and their representations for a variety of reasons. First, the CSG representation from which the BRep is to be derived describes an r-set whose boundary may not be 2-manifold. We wish to guarantee that any solid representable in CSG can be represented using a BRep data structure capable of capturing all adjacency relationships (nonmanifold and otherwise) implied in the CSG tree.

Second, even though a nonmanifold solid may not be strictly manufacturable, such a solid may be created in passing by the designer as an intermediate stage of the modeling process. A subsequent modeling operation may add or eliminate material, generating a solid whose boundary is 2-manifold.

Third, it is convenient to generate and manipulate nonmanifold solids internally during the course of the boundary evaluation algorithm, for example when processing certain types of edge coincidences. Finally, it is common to have a 3-manifold solid whose boundary is not everywhere 2-manifold (cf., the hatched portion of the boundary in Figure 2(a)).

There are also a number of reasons for seeking a representation and boundary evaluation algorithm supporting partially bounded solids. First, CSG representations are ultimately general Boolean combinations of half-spaces. Most CSG-based systems only provide users with bounded primitives, but this is an artificial restriction. CSG representations will describe partially bounded solids if the half-spaces along with the Boolean operations applied to them do not enclose a finite volume of space. As we argued above for nonmanifold solids, we wish to be able to describe in the BRep whatever is described in the CSG tree; hence we seek a BRep data structure capable of representing partially bounded solids and a boundary evaluation algorithm capable of accepting and producing these unbounded representations of solids.

A second closely related motivation is that it becomes almost trivial to provide "user-defined primitives" (sometimes called "superprimitives") in a very natural way. Observe that if the boundary evaluator supports partially bounded solids, then there is no need internally for BRep templates for standard bounded primitives. That is, if a system developer wished to supply a standard set of simple bounded modeling primitives such as blocks and bounded cylinders, the instancing parameters could be immediately converted internally into the appropriate Boolean combination of half-spaces, and the boundary evaluator could then be invoked on this subtree. In this scenario, the bounded primitives are essentially viewed by the system as a macro since the internal algorithms know nothing of entities other than half-spaces and Boolean operations. Of course the macro definition could be saved and associated with the appropriate tree node, but this would only be used by the user interface to expedite subsequent design modifications.

Now if a user wishes to extend the system by adding specialized modeling primitives commonly used in some application, he need only create a specialized set of macros. The resulting user-defined primitives could be made to look internally exactly like the standard system-supplied primitives. Note that this also allows user-defined primitives to be represented with much smaller subtrees than would be possible by defining them in terms of standard system-supplied bounded primitives. This would help to minimize storage and computational requirements when using these primitives.

We have demonstrated the feasibility of this user-defined primitive mechanism by defining a general "convex rounded pocket" primitive which we used to generate the pockets in the familiar CAM-I ANC101 test part.

A third motivation for the use of partially bounded solids relates to certain common modeling and display processes. When defining through holes, slots, chamfers, and the like, it should be unnecessary for a designer to specify artificial limiting geometry. For example, there should be no need to provide a starting and stopping point for a cylinder which is to be used to drill a hole all the way through a part. Similarly, if a designer needs to generate a sectioned drawing of a part, we should not require a fully bounded solid to be defined and intersected with the part when in fact the designer is only concerned with a few critical sectioning surfaces (cf. Figure 9).

We shall not discuss in this paper the algorithms for the basic geometric computations employed at a low level in the system. Specific details related to curve and surface representations as well as analytical algorithms such as intersections, the computation of differential quantities of curves and surfaces, and the determination of coincidence relationships between points, curves and surfaces are transparent to the boundary evaluation algorithm. This algorithm makes appropriate requests through a black-box interface but has no knowledge of curve and surface representations or geometric algorithms. (We confess to one exception in Section 5.4 for which we do not yet have a good answer, however.) We have implemented, tested, and evaluated the boundary evaluation algorithm described here using a set of geometric representations and algorithms, the majority of which are described in [1, 10-12].

The remainder of this paper is organized as follows. In Section 2, we review previous work in this area. Section 3 describes relevant aspects of the host system. We present in Section 4 the modifications to the basic algorithm described in [13] which are required to support nonmanifold input and output solids. Similarly, Section 5 describes those modifications necessary to support partially bounded solids. Finally, we summarize in Section 6.

2.0 Previous Work

In [13] we surveyed a number of papers which described various algorithms for BRep-CSG conversion for traditional manifold and bounded solids [2-4, 6-9, 16]. Somewhat less has appeared dealing with nonmanifold boundary evaluation, and to our knowledge nothing has appeared on boundary evaluation for partially bounded solids in the sense that we mean.

Hoffmann, et. al. discuss the need to order line segments about a shared vertex on a plane and polygonal faces about a shared edge in space in order to represent properly nonmanifold faces and solids. [7] The emphasis in that paper is on the complex issue of numerical reliability. The argument is well-made that even within the relatively simple domain of planar polyhedra, ensuring robustness requires employing a mixture of numerical techniques and organizing the algorithm properly so as to minimize the possibility of generating inconsistent answers to related but separate queries. The emphasis in this paper is not numerical reliability, rather on how an existing algorithm can be enhanced to support nonmanifold and partially bounded solids. We do make a few general observations on numerical reliability later, but we attempt no exhaustive treatment of this important issue here.

Rossignac and Requicha propose a modified constructive representation for describing objects of mixed dimensionality [18]. The emphasis in their work is the representation of dimensionally non-homogeneous objects which may have complex internal structure.

Boundary evaluation algorithms for polyhedral objects of mixed dimensionality have been described in [5, 14]. The general approach involves a bottom-up intersection and incidence testing phase. followed by a top-down merging operation.

3.0 Background

The algorithm to be described has been implemented in the geometric modeler cryph being developed and used as a research and teaching tool at the University of Kansas. Cryph maintains a dual CSG-BRep representation of solids in which the two representations are intimately linked. For example, we can directly determine the set of faces which lie on a given half-space in the CSG tree, and we can directly query the half-space in the CSG tree on which a given BRep face lies.

The CSG representation is stored as an n-ary tree with intersection, union, and difference operators at non-terminal nodes and unbounded half-spaces at the leaves. A BRep can be associated with each nonterminal node of the tree, although in practice usually only the top level BRep is maintained.

The BRep is based on Weiler's Radial Edge data structure [19]. Among other things, this data structure explicitly separates the concept of a *use* of a topological element from the element itself. As we shall see, this separation is a natural match to the way we think about boundary evaluation, and it allows the logic in critical sections of the algorithm to be simplified. We have extended it somewhat for this work to describe unbounded and partially bounded edges. For example, an unbounded edge on a straight line is represented as a self edge loop on the line. Further details on these modifications are provided in Section 5.1.

Edge uses are oriented so that the interior of a face is on the left as one walks along the edge use. Given two edge uses *eu1* and *eu2* of a face f lying on a surface s, writing *eu1-eu2* indicates that *eu1* stops at the vertex where *eu2* starts. *eu1-eu2* defines an oriented curvilinear wedge (or "sector") on *s* inside of which *f* lies. The curvilinear wedge is called the neighborhood defined by *eu1-eu2* and is written as *N(eu1,eu2)*. Each edge use, *eu*, points to the use on the immediately adjacent face (radial(*eu*)) as well as the use on the other side of the face to which it belongs (mate(*eu*)).

Connected faces are bounded by one or more compatible loops of adjacent edge uses. Two loops on a surface are said to be *compatible* if there is a path on the surface from an edge use on one loop into its local neighborhood which arrives at an edge use of the other loop on the side of its local neighborhood such that the path does not cross any edge in either loop. On surfaces of genus zero, two compatible loops will always form a well-defined (but possibly unbounded) connected face. Though symmetrical, compatibility is not a transitive relationship.

An edge lying on one of the solids input to the boundary evaluation algorithm is called a *self edge (SE)*. A new edge arising from the intersection of a face on one operand with a face of the other is called a *cross edge (CE)*. If a CE is also an SE, we call it a *CESE*. We shall use "pure CE" or "pure SE" when describing edges which are not CESEs.

It is well known that it suffices to implement the regularized Boolean intersection operation if we have a unary complement operator. Using the Radial Edge data structure [19], the complement operation requires only pointer modifications, hence this adds little overhead to the overall algorithm. We therefore develop the algorithm assuming only regularized intersections are being performed, but for clarity we use union, intersection, and difference operations when showing sample geometry in figures.

Our basic algorithm for computing the solid C from the intersection of given solids A and B is based on the standard generate-and-test paradigm: Generate sets of faces, edges, and vertices known to contain all those of C, then test each member of the three sets to determine which ones belong to the boundary of C [16]. It is both sufficient and computationally expedient to focus on edges, inferring faces and vertices in the process. If an edge is completely surrounded either by solid material or by air, then it is not on the boundary; otherwise it is. Explicit edge classification is the process of characterizing this neighborhood of an edge and subsequently concluding whether it is part of the boundary of C.

Explicit edge classification is an expensive process, hence we have developed our algorithm so as to avoid the vast majority of the edge classifications which would normally be required. We need only resort to explicit determination and analysis of edge neighborhoods for CESEs [13].

The essence of our basic algorithm for computing ($C = A \cap B$) proceeds as follows. More complete details are presented in [13].

1. If A and/or B is the empty solid, then generate C trivially and exit.
2. Partition SEs of A at their points of intersection with faces of B.
3. Partition SEs of B at their points of intersection with faces of A.
4. Compute and partition CEs, associating uses with surfaces on A and B.
5. Classify CESE uses discovered during step 4.
6. Infer SE classifications, splitting and merging faces as appropriate.
7. Check for split and/or merged shells.

The remainder of the paper focuses on how this basic algorithm must be extended for nonmanifold and partially bounded solids.

4.0 Nonmanifold Issues

Portions of steps 4-6 of the basic algorithm reviewed in the previous section must be extended to allow nonmanifold input and output solids. Following a treatment of these extensions, we conclude by discussing nonmanifold vertex conditions. We shall see that some will be automatically represented without any additional effort, while others must be explicitly sought.

4.1 Cross Edges on Tangent Intersection Curves

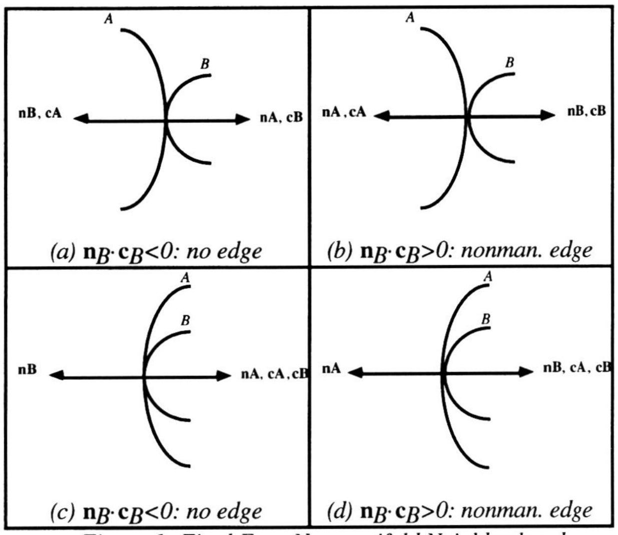
An edge arising from the transverse intersection of a face from A with one from B will be partly surrounded by material and partly surrounded by air (and hence will lie on C) regardless of the orientations of the two faces involved. However if the edge lies on a tangent intersection curve (i.e., one along which the two underlying surfaces have parallel normal vectors), then special consideration is required. If the input and output solids are known to be manifold, such intersections can be ignored [13]. Otherwise a cross edge is required on a portion of such a tangent intersection curve only if one of the following is true:

- A self edge occupies this interval of the curve. A CESE is recorded in this case, and modifications to the CESE sector analysis operation necessary to deal with this are discussed in Section 4.2.
- A nonmanifold edge will result. The logic required to determine if a new edge should be created (i.e., to determine if the result is locally nonmanifold) requires examination of local differential properties of the surfaces involved and is discussed in the remainder of this section.

The portion of the algorithm charged with deciding if a new edge should be created is given a particular interval on an intersection curve to consider. The presence of a CESE on the given interval, if one existed, would have been detected by the algorithm prior to this stage. We are therefore free to assume here that no edge on either input solid occupies this portion of the curve. The logic required to determine if a new edge should be created is essentially the same as that used for CESE sector analysis, but it can be greatly simplified since we know that no existing edges lie on the curve. That is, the neighborhoods of the tentative new CE with respect to A and B are determined solely by the local surface geometry as modified by face use orientation flags.

We compute the outward pointing face use unit normal vectors $\mathbf{n}_A$ and $\mathbf{n}_B$ at the midpoint of the curve segment and then slice both surfaces with a plane passing through the midpoint and perpendicular to the curve tangent at the point. We compute certain local differential properties of the plane section curves, most notably $(\mathbf{c}_A, \kappa_A)$ and $(\mathbf{c}_B, \kappa_B)$, the unit curvature direction vectors and magnitudes. Observe that at most one of the curvature magnitudes can be zero. Without loss of generality, we can assume that the roles of A and B have been assigned based on these local differential properties. In particular, we can assume that $\kappa_B \geq \kappa_A$. In the case that $\kappa_B = \kappa_A$, then either ($\mathbf{c}_A \cdot \mathbf{c}_B < 0$) in which case it does not matter which assumes the role of B, or higher order derivatives can be employed in order to insure that B is the one which is bending the most at the common point. It is not difficult to see that a nonmanifold edge is only possible when the outward pointing face use normals are oppositely directed. This observation and the results of an exhaustive consideration of the remaining four possibilities (see Figure 1) leads to the following simple pair of tests.

```
if $\mathbf{n}_A \cdot \mathbf{n}_B < 0$ then
    if $\mathbf{n}_B \cdot \mathbf{c}_B > 0$ then
        create a nonmanifold edge
```



(a) $\mathbf{n}_B \cdot \mathbf{c}_B < 0$*: no edge* *(b)* $\mathbf{n}_B \cdot \mathbf{c}_B > 0$*: nonman. edge*
(c) $\mathbf{n}_B \cdot \mathbf{c}_B < 0$*: no edge* *(d)* $\mathbf{n}_B \cdot \mathbf{c}_B > 0$*: nonman. edge*

Figure 1: Final Four Nonmanifold Neighborhood Possibilities

In all other cases, the neighborhood of the current portion of the tangent intersection curve is either empty, full, or occupied by only one of A or B. Therefore no edge should be created in any of those other cases.

Once we have decided that a nonmanifold edge is to be created, four edge use pairs are generated, two pairs for each of the two surfaces involved. The two edge use pairs assigned to a given surface are oppositely oriented. All that remains is to ensure that these new nonmanifold CEs are properly handled during the subsequent classification inference step. This will be discussed in Section 4.3.

A final related consideration involves the process of delimiting shells in step 7 of the basic algorithm. We wish to consider sets of faces which touch only along nonmanifold edges as separate shells. When visiting adjacent faces during delimiting of shells, we therefore wish to follow adjacencies by passing through solid material instead of air. That is, when at an edge use *eu*, we select the face containing mate(radial(mate(*eu*))) instead of the one containing radial(*eu*).

4.2 CESE Sector Analysis

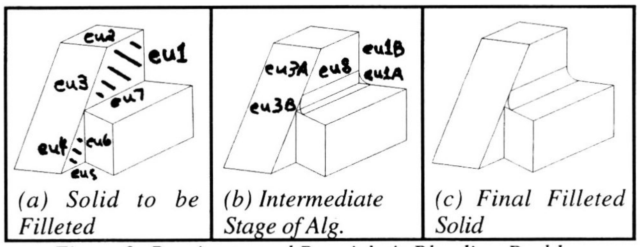
Recall that we classify a CESE by slicing both input solids with a plane passing through the midpoint of the CESE. If one or both of the input solids is nonmanifold along the CESE being examined, the required bookkeeping is fairly elaborate. Fortunately the nonmanifold problem can be reduced to a series of manifold ones. That is, given the ordered list of sectors around A and the ordered list of sectors around B:

```
for each sector sB in B's sector list do
    for each sector sA in A's sector list containing onlyA do
    • combine sA and sB
    • replace sA with those resulting sectors containing onlyA or AandB
    • replace sB with those resulting sectors containing onlyB
```

Once this has been accomplished, the sectors in A's list are quasi-disjoint from those in B's list, and the two ordered lists can be merged. Those sectors in the final merged list which contain *AandB* bound portions of the possibly nonmanifold result, and edge use pairs are retained and/or created at the boundaries of these sectors. The appropriate radial ordering of the edge use pairs is determined by the ordering in the final merged list.

4.3 Classification Inference

We rely heavily on classification inference in the algorithm, and this operation is probably the most sensitive to nonmanifold conditions. When more than two edges share a vertex on a face, there are multiple valid ways to connect the edge uses. Consider the example illustrated in Figure 2 where we are applying a fillet to a simple part.



(a) Solid to be Filleted *(b) Intermediate Stage of Alg.* *(c) Final Filleted Solid*

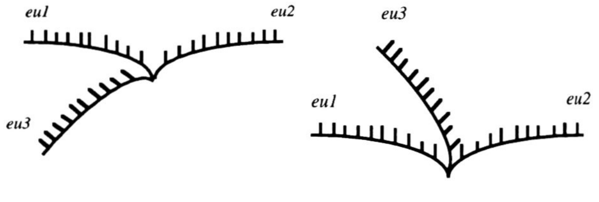
Figure 2: Rossignac and Requicha's Blending Problem

The geometry of Figure 2 appeared in [17] as an example of a blend with a complex end condition. Here we focus on the hatched surface of Figure 2a and consider the possible adjacency relationships among the indicated edges. There are three options: (i) a single maximal face bounded by a single loop: *eu1-eu2-eu3-eu4-eu5-eu6-eu7*, (ii) a single maximal face bounded by two loops touching at a vertex: *eu1-eu2-eu3-eu7* and *eu4-eu5-eu6*, or (iii) two connected faces: one bounded by *eu1-eu2-eu3-eu7* and one bounded by *eu4-eu5-eu6*. Any one of these three choices is valid in that they unambiguously define the portions of the plane forming part of the boundary of the solid.

In Figure 2b, we have positioned a solid to be used as a fillet and consider an intermediate stage of the boundary evaluation algorithm invoked in order to form the union of the part with the solid fillet. Edge uses *eu1* and *eu3* have been partitioned into *eu1A*, *eu1B*, *eu3A*, and *eu3B*. The algorithm determines that the new edge use *eu8* connects so that we get *eu3A-eu8-eu1B*, and the edge inference mechanism then deduces that all edges between the attachment points must be killed (i.e., *eu3B-...-eu1A*).

The correct result in Figure 2c is generated only if the input topology is represented as described in (ii) or (iii) above. Edges *eu4*, *eu5*, and *eu6* would be incorrectly deleted if the adjacencies of option (i) had been generated for the initial object.

The generation of connected faces by the boundary evaluation algorithm is a sufficient condition for the proper operation of the classification inference logic. When multiple edge adjacencies are possible at a given vertex, we ensure that connected faces will be generated by establishing edge adjacencies which minimize the size of the neighborhood determined by pairs of edge uses. Referring to the example of Figure 3, suppose we generate *eu1-eu2* on a face, and we then discover that edge use *eu3* also stops at their common vertex. If *eu1* lies in *N(eu3,eu2)* as in Figure 3a, then we preserve *eu1-eu2*. If instead *eu3* lies in *N(eu1,eu2)* as in Figure 3b, we generate *eu3-eu2*.



(a) eu1 in N(eu3,eu2) *(b) eu3 in N(eu1,eu2)*
Figure 3: Determining Edge Use Adjacencies

The computational tools required are similar to those described for CESE analysis: curves on a surface must be ordered about a common point based first on tangents, then on higher order derivatives.

Once the mechanism for determining proper edge use adjacencies has been established, the loop compatibility scheme for generating connected faces can be used exactly as described in [13]. For example, we determine that the CE on the closed curve on the shaded face of Figure 4 should connect before and after SEs lying on the ellipse which it touches, but that the edges on the other half of the ellipses should belong in a separate loop defining a separate connected face.

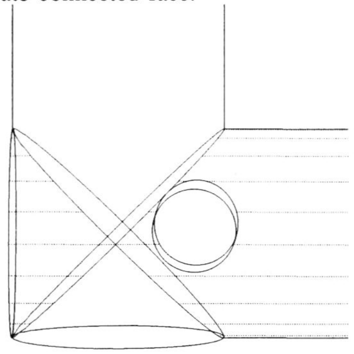
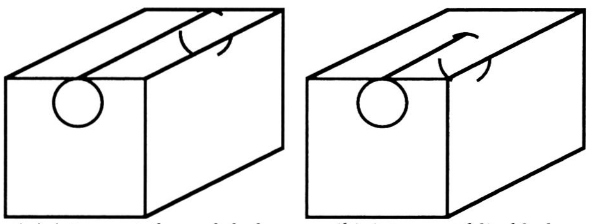


Figure 4: Nonmanifold face conditions on the union of two bounded right circular cylinders with a protruding pin.

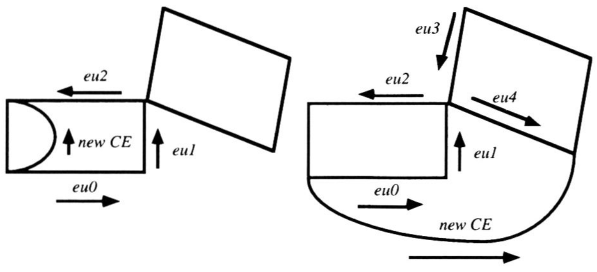
In section 4.1 we saw that two oppositely-oriented uses of the same nonmanifold edge can be assigned to a surface. Such pairs will be assigned to the same loop (and hence connected to one another) or to different loops based on how they relate to the other edge uses on the surface. The process of minimizing the size of neighborhoods as just discussed produces the correct connectivities. Consider the example of Figure 5a in which the cylindrical through hole is tangent to the top face of the block. The two uses of the nonmanifold edge will be assigned to the same loop on the cylindrical face, but to different loops (and different connected faces) on the top plane of the block. If the cylinder went only part way into the block as in Figure 5b, then the two uses on the top plane would be adjacent to each other in the same loop.



(a) A tangent through hole *(b) A tangent blind hole*
Figure 5: Nonmanifold Edges

When input faces are manifold, we are guaranteed that open strings of new CE uses will hit loops an even number of times. In each pair of hits, there is one where a CE use connects after an existing SE use, and one where a CE use (possibly the same one) connects before an existing SE use. The two SE uses will be different if the result is locally manifold, but may be the same if not.

However when input faces are nonmanifold (i.e., when more than two edges share a vertex on the underlying surface as in Figure 6), it is possible for an open string of CE uses to bridge two loops, each loop being hit by a string of CE uses only once. If this situation arises, the classification inference mechanism must know to jump from one loop to another when deleting SE uses. The trick is to detect and handle this as efficiently as possible. It does not happen often, so we do not want to add lots of overhead to the algorithm and data structures if we can avoid it.



(a) Delete eu0, eu1, eu2 *(b) Delete eu0, eu1, eu4*
Figure 6: Processing nonmanifold vertices

Recall that we infer OUT classifications for SE uses by traversing edge uses in loops between connection points. During this traversal, we determine at each vertex if the vertex is nonmanifold on the surface. If so, we must explicitly classify the next SE use which would normally have been deleted without any explicit classification. This will be a trivial classification, however, since the use either lies on a CESE (in which case its classification is already known), or it lies on a pure SE in which case a simple point-CSG tree classification suffices. That is, we can classify a point in the interior of the SE with respect to the CSG tree describing the other operand. The result of this classification is the classification of the SE with respect to *C*.

As an example, consider the situations depicted in Figure 6. In Figure 6a, we have added the indicated new CE, and we are now in the midst of the subsequent inference and deletion phase. We have just deleted edge use *eu1*, and we are about to delete *eu2*. Since their common vertex is nonmanifold, we explicitly classify *eu2*, verify that it is to be deleted, and then proceed with the deletion process. Now consider the variation in Figure 6b. Classification of *eu2* tells us that it is to survive. Checking other edge uses leaving the common vertex, we find that *eu4* (in a different loop and face) is the next edge use to be deleted. We connect *eu2* to the predecessor of *eu4* (i.e., *eu3*), and proceed with deletion by killing *eu4* and its successors. If there are multiple choices for *eu4*, neighborhood analysis as discussed above allows us to pick the right one. This operation could be optimized by maintaining tags in vertex uses which would indicate whether the use is nonmanifold on the surface on which it lies instead of determining this condition algorithmically by using vertex-edge adjacency queries.

4.4 Nonmanifold Vertices

Candidate nonmanifold points are identified during edge partitioning (steps 2 and 3 of the basic algorithm) and during cross edge generation (step 4). In most cases, no extra effort is required on the part of the boundary evaluation algorithm to capture the nonmanifold vertex condition. The normal operation of the algorithm will simply result in a vertex shared by locally distinct connected solids touching only at the nonmanifold vertex.

For example, consider the edge partitioning step when an edge *e* from *A* intersects a face *f* from *B* at a single point *P*. *P* may be coincident with a vertex of *f*, it may lie in the interior of an edge of *f*, or it may lie strictly in the interior of *f*. If *P* is at a vertex of *f*, *e* will be split into two edges, both of which share this vertex of *f*. If instead *P* lies in the interior of an edge of *f*, a new vertex will be created at this point and used to split both *e* and the edge of *f* involved. In either event, if this vertex turns out to be nonmanifold on *C*, the information recorded in the BRep during this process will indicate this.

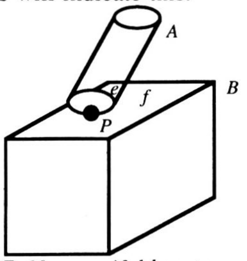


Figure 7: Nonmanifold vertex on a face

If *P* lies strictly in the interior of *f*, however, additional work is required in the event the vertex turns out to be nonmanifold. (See, for example, Figure 7.) Either *P* is already a vertex *v* of *e* or we create a new vertex at *P*, call it *v*, and split *e* at *v*. We remember (*v*,*f*), and add *v* to *f* as a nonmanifold single vertex loop if both *e* and *f* survive on *C* and if no other edge on *f* references *v*.

During cross edge generation (step 4 of the basic algorithm), we may discover a pair of surfaces which are tangent at one or more isolated points. These points may indicate nonmanifold conditions on the result. In a manner similar to that just discussed for SE partitioning, we can simply keep track of the point(s) of tangency (there is no need to create vertices yet) and the two surfaces involved. At the end of the algorithm, we can determine if nonmanifold single vertex loops should be created for such a point by determining if it lies in the interior of a face on each surface.

When creating the preliminary minimal BRep on an unbounded surface, the standard approach is to create one shell with one face bounded by one single vertex loop. If the surface has a nonmanifold point, however, two or more such shells are generated which touch at a nonmanifold vertex lying at the nonmanifold surface point. The obvious example is a right circular cone in which two minimal shells are created.

5.0 Unbounded Issues

We begin by describing our extensions to the Radial Edge data structure which allow it to describe unbounded and partially bounded edges. Then we describe how classification inference is affected in loops with such edges. Next we describe how classification of points with respect to faces is affected when faces might have infinite area. Finally we discuss a difficult numerical issue which can arise on such faces.

5.1 Data Structures

The primary issue is the representation of unbounded and partially bounded edges. No special data structure consideration is required for partially bounded faces or partially bounded solids in the BRep.

It is common to maintain loops of edge uses as doubly-linked circular lists. Having both "next" and "previous" pointers in this list is of course redundant, but it is common since it allows frequently occurring edge manipulation operations to be performed very efficiently.

Edge uses occupying an entire closed curve are self-edge loops; that is, their next and previous pointers are self-referential. We simply extend the notion of self-edge loops for unbounded and partially bounded edges. An edge occupying an entire straight line, for example, is simply a self-edge loop on the line. A loop referencing such an edge may be used to define a face occupying half of a plane.

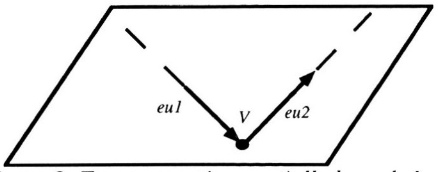


Figure 8: Two connecting partially bounded edge uses

If an edge is bounded at one end (connecting to another edge use there) but unbounded at the other end, then a straightforward extension of this idea is used. Suppose two lines in a plane intersect at a distinct point, and consider the two rays on the lines determined by the intersection point (Figure 8). Edge use *eu1* starts at infinity on the first line and stops at vertex *V*; edge use *eu2* starts at *V* and ends at infinity on the other line. The loop defined by *eu1-eu2* determines a face covering a quarter of the plane. The "previous" pointer for *eu1* is *eu1*, and the "next" pointer for *eu2* is *eu2*. Clearly this notion extends to loops defined by *n* edge uses *eu1-...-eun* where *eu1* and *eun* are partially bounded, and the others are fully bounded. Note that having both "next" and "previous" edge use pointers is no longer redundant using this scheme.

Two unbounded edges on straight lines in different loops on a plane can define a face occupying an infinite strip of the plane. For example, Figure 9 illustrates an L-Bracket being sectioned by portions of five planes. The five edges at the top and bottom of the planes are for visualization purposes only. The only actual edges on the sectioning geometry are the four unbounded vertical edges.

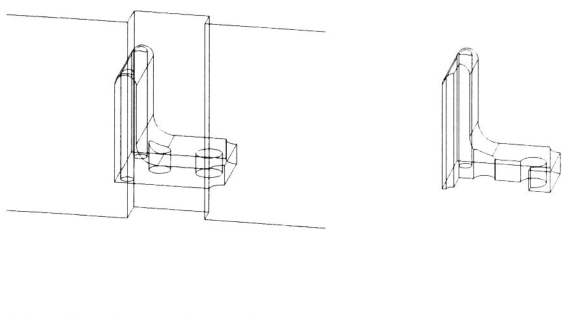

(a) L-Bracket with sectioning geometry consisting of portions of 5 planes *(b) L-Bracket after the sectioning operation.*

Figure 9

5.2 Classification Inference

Recall that the basic classification inference scheme works by deleting SE uses in loops between points at which CE uses are attached. This involves remembering the SE use immediately following an SE after which a new string of CE uses is attached. In the example of Figure 10a, we attach the string of new CE uses *CEu1, ... , CEuk* so that *CEu1* follows *eu(j-1)* and *CEuk* precedes *eu(i+1)*. In the process, we remember *euj* as the first SE use in a string of uses whose OUT classification will be inferred. After all new strings of CE uses have been added to the face, we begin deleting strings of edge uses beginning with each saved *euj* and stopping with the edge use prior to one to which a new CE has been connected. In the example of Figure 10a, edge uses *euj*, ..., *eun*, *eu1*, ..., *eui* will be deleted.

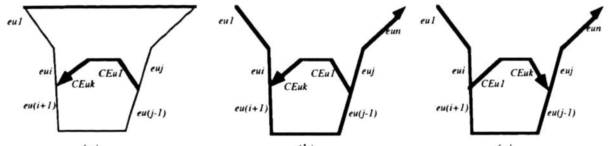


Figure 10: Classification inference with unbounded edges

If instead the face is unbounded as in Figure 10b, additional action is required since we can no longer get from *eun* to *eu1* during the edge use deletion operation. An obvious solution is to detect that the first edge use of a loop is unbounded and then remember *eu1* at the same time we remember *euj*. However this is not a general solution as it will only work for simple situations such as that depicted in Figure 10b. If the new string of CE uses is also unbounded, for example, then we will not want to delete the string of edge uses headed by *eu1*. Alternatively if the new string of CE uses is oppositely oriented as in Figure 10c, then again we do not wish to delete the string of uses starting at *eu1*, and in fact the standard inference mechanism as originally described generates the correct result.

The "obvious solution" is almost the correct solution. We check to see if the first edge use of a loop into which a string of CE uses is added is unbounded. If so, we explicitly classify this leading unbounded edge use and remember it only if it is to be deleted. Just as with the explicit classification we required in Section 4.3, this classification adds negligible overhead. The edge use will either lie on a CESE, in which case its classification is already known, or it can be classified by a simple point-CSG tree classification.

5.3 Point Classification

Classification of points with respect to edges and related utilities such as generating points in the interior of edges are trivial for edges which are unbounded. For edges which are partially bounded, the logic is somewhat more complex than that for normal fully bounded edges, but fairly straightforward extensions to the usual logic can be implemented to deal with them.

Classification of points with respect to faces is more difficult. The traditional approach is to shoot a ray on the surface from the point and either (i) count intersections with edges or (ii) compare the ray and edge use orientations at the closest edge use encountered. Neither of these approaches work properly when faces are allowed to be unbounded since one cannot conclude that a point is outside a face if the ray intersects no edges at all. One approach is to continue to generate and shoot rays until one finally hits an edge on a face. This can be expensive, however, since arbitrary rays on a surface require more complex intersection logic.

(If faces lie only on natural quadrics, the rays can be restricted to lines and circles. Clearly the rays can always be circles on spheres. For cones and cylinders, choose as the initial ray a straight line. If it hits no edges, shoot a circular ray. If this also misses, we know that all points on the circle have the same classification as the original point, so we can proceed to shoot straight line rays through other points on this circle until we hit something. This rarely requires more than a couple of rays.)

Rays on faces pass through vertices or are coincident with edges a surprisingly large number of times. This is probably due to the fact that we tend to construct regular shapes in regular orientations, and we often need to classify points which have fairly regular positions with respect to these vertices and edges. A related problem arises during display in that a surprisingly large number of silhouette curves pass through vertices and edges on surfaces, again the predictable result of employing regular orthographic views of models. Different rays can be used for point classification, but as just noted, this can be expensive, and it does not solve the silhouette problem since the silhouette curve is not arbitrary. Therefore we attempt to determine an answer in these cases by explicitly examining vertex neighborhoods on surfaces. Our examination of vertex neighborhoods fails only if we cannot locally orient the curve (e.g., at a nodal or cuspidal point on a space curve) or the surface (e.g., at the vertex of a cone).

A promising approach for the general problem of classification of points with respect to faces is to develop methods for exploiting the presence of the dual CSG representation. What is required is a scheme for extracting from the CSG representation of the solid a CSG description for connected faces. General classification of points with respect to faces could then be reduced to the much simpler problem of classifying points with respect to CSG trees, and we would be able to finesse the problematic issues raised in the preceding paragraphs of this section.

5.4 Numerical Issues Specific to Unbounded Geometry

While the general issue of numerical reliability is not a focus of this paper, the use of unbounded half-spaces occasionally introduces difficult numerical problems which would not otherwise arise. We discuss such an example here and then make a few general remarks on numerical reliability in the summary.

Consider a variation of the "convex rounded pocket" primitive mentioned in Section 1 which allows a draft angle to be specified. The internal representation differs only in that right circular cones are used instead of cylinders (see Figure 11). When the boundary evaluation algorithm is evaluating the solid corresponding to the intermediate union node, edges lying on cone rulings determined by plane-cone intersections meet edges lying on the cone-cone intersection curves at points very far away from the region occupied by the pocket itself. These connection points must be properly detected and represented even though they do not lie on the final solid. The edge classification inference logic which will be used when evaluating the final intersection node depends on proper connectivity of edges having been generated for this intermediate solid. In order to generate these connectivities properly, the algorithm currently relies in part upon classification of points with respect to faces. Differences between coordinates of points involved can easily be ten or more orders of magnitude, however, and it becomes impossible to find ray-edge intersection points reliably using normal methods.

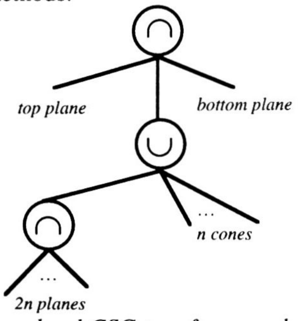


Figure 11: Three-level CSG tree for user-defined primitive

In order to avoid the use of symbolic computations or other mechanisms for dramatically increasing the number of significant digits, we have worked around this problem for now by employing cone-specific logic. This is of course a crude hack which we must replace with a more general and consistent solution. One possibility is to employ the CSG representation of connected faces as mentioned in the previous section, thereby completing avoiding the normal method for classifying points with respect to faces.

6.0 Summary

We have described how our earlier boundary evaluation algorithm [13] which depended heavily upon inference of edge classifications can be extended to support solids which are nonmanifold and/or only partially bounded. We required explicit edge classifications in a couple of additional situations, but these classifications added negligible overhead to the algorithm since the edge was either a CESE, in which case its classification was already known, or it was a pure SE, in which case a simple classification of a point with respect to the dual CSG tree of one of the input operands sufficed.

We also identified some numerical difficulties associated with the low-level point-with-respect-to-face classification query. These difficulties provide motivation for the study of how CSG representations for connected faces can be derived directly from CSG representations of input solids since such representations would allow us to finesse these difficult problems.

The focus of this work has been to consider how boundary evaluation algorithms based on edge classification inference could be extended to handle nonmanifold and partially bounded solids. We have not attempted an exhaustive analysis of numerical reliability since this is largely an orthogonal issue. Numerical issues generally fall into two categories. The first relates to how reliably curves and surfaces can be intersected, especially in the presence of tangencies and other singularities. These issues are dealt with at a low level in curve and surface intersection and analysis packages. In our system, we employ a set of tools designed primarily using vector geometry techniques, and numerical reliability was a central consideration throughout the development of those tools [1, 10, 11, 12].

The other category of numerical issues involves queries posed independently, but which yield logically inconsistent answers. For example, at one stage of the algorithm we may conclude that two planar faces are coincident, but later conclude inconsistently that two linear edges on those faces are skew. We have not attempted an analysis of how well our algorithm operates with respect to these sorts of issues. We have evaluated reliability empirically in a number of ways. For example, we have used the well-known benchmark of intersecting a cube with an identical one rotated by small amounts about the x-, y-, and z-axes. Given a unit cube centered at the origin, our algorithm generates the correct intersection for rotations down to 0.26 degrees. Below this the algorithm fails. We tried an analogous test with a unit bounded cylinder centered on the origin. Our algorithm produced the correct results down to 0.19 degrees, failing at angles below that threshold. We believe that techniques for avoiding the sorts of inconsistent queries which give rise to the algorithmic failures mentioned could be added to this algorithm in a way which does not impact the methods which form the thrust of the work described here, namely the ability to evaluate and represent the BRep for nonmanifold and partially bounded solids.

Acknowledgments

The author is indebted to Kevin Weiler for his willingness during the early stages of this work to discuss at length various aspects of his Radial Edge data structure, particularly how representations of unbounded and partially bounded edges might best fit into the overall philosophy of the data structure. This work was supported in part by the University of Kansas General Research Allocations #3143-X0-0038 and #3270-X0-0038.

References

1. E. Chionh, R. N. Goldman, and J. R. Miller, Using Multivariate Resultants to Find the Intersection of Three Quadric Surfaces, *ACM Transactions on Graphics*, Vol. 10, No. 4, October 1991, pp. 378-400.
2. H. Chiyokura, *Solid Modelling with DESIGNBASE: Theory and Implementation,* Addison-Wesley, 1988.
3. G. A. Crocker and W. F. Reinke, Boundary Evaluation of Non-Convex Primitives to Produce Parametric Trimmed Surfaces, *Computer Graphics* (Proceedings SIGGRAPH '87), Vol. 21, No. 4, July 1987, pp 129-136.
4. G. A. Crocker and W. F. Reinke, An Editable Nonmanifold Boundary Representation, *IEEE Computer Graphics and Applications*, Vol. 11, No. 2, March 1991, pp 39-51.
5. E. L. Gursoz, Y. Choi, and F. B. Prinz, Boolean set operations on non-manifold boundary representation objects, *Computer-Aided Design*, Vol. 23, No. 1, January/February 1991, pp. 33-39.
6. C. M. Hoffmann, *Geometric & Solid Modeling: An Introduction,* Morgan Kaufmann Publishers, Inc, 1989.
7. C. M. Hoffmann, J. E. Hopcroft, M. S. Karasick, Robust Set Operations on Polyhedral Solids, *IEEE Computer Graphics and Applications*, Vol. 9, No. 6, November 1989, pp. 50-59.
8. M. Mantyla, Boolean Operations of 2-Manifolds Through Vertex Neighborhood Classification, *ACM Transactions on Graphics,* Vol. 5, No. 1, January 1986, pp 1-29.
9. M. Mantyla, *An Introduction to Solid Modeling,* Computer Science Press, Rockville, Maryland, 1988.
10. J. R. Miller, Geometric Approaches to Nonplanar Quadric Surface Intersection Curves, *ACM Transactions on Graphics*, Vol. 6, No. 4, October 1987, pp. 274-307.
11. J. R. Miller and R. N. Goldman, Using Tangent Balls to Find Plane Sections of Natural Quadrics, *IEEE Computer Graphics and Applications*, Vol. 12, No. 2, March 1992, pp. 68-82.
12. J. R. Miller and R. N. Goldman, Geometric Algorithms for Detecting and Calculating All Conic Sections in the Intersection of Any Two Natural Quadric Surfaces, *Computer Vision, Graphics, and Image Processing*, Vol. 57, No. 1, January 1995, pp. 55-66.
13. J. R. Miller, Incremental Boundary Evaluation Using Inference of Edge Classifications, *IEEE Computer Graphics and Applications*, Vol. 13, No. 1, January 1993, pp. 71-78.
14. M. Muuss and L. Butler, Combinatorial Solid Geometry, Boundary Representations, and n-Manifold Geometry, in *State of the Art in Computer Graphics: Visualization and Modeling*, Rogers and Earnshaw, ed., Springer Verlag, New York, 1991, p. 368.
15. A. A. G. Requicha, Representations for Rigid Solids: Theory, Methods, and Systems, *ACM Computing Surveys,* Vol. 12, No. 4, pp. 437-464, December 1980.
16. A. A. G. Requicha and H. B. Voelcker, Boolean Operations in Solid Modeling: Boundary Evaluation and Merging Algorithms, *Proceedings of the IEEE*, Vol. 73, No. 1, January 1985, pp. 30-44.
17. J. R. Rossignac and A. A. G. Requicha, Constant-Radius Blending in Solid Modeling, *Computers in Mechanical Engineering,* Vol. 3, No. 1, July 1984, pp. 65-73.
18. J. R. Rossignac and A. A. G. Requicha, Constructive Non-Regularized Geometry, *CAD*, Vol. 23, No. 1, Jan/Feb 1991, pp. 21-32.
19. K. J. Weiler, *Topological Structures for Geometric Modeling*, Ph.D. Dissertation, Computer and Systems Engineering, Rensselaer Polytechnic Institute, August 1986.