Maintaining Topology in Geometric Descriptions with Numerical Uncertainty

Mark G. Segal and Carlo H. Séquin *
Computer Science Division
Department of Electrical Engineering and Computer Science
University of California
Berkeley, CA 94720

June, 1988

Abstract

Algorithms for computer graphics or computational geometry often infer the topological structure of geometrical objects from numerical data. Unavoidable errors (due to limited precision) affect these calculations so that their use may produce ambiguous or contradictory inferences.

An object description associating a tolerance with each of its topological features (vertices, edges and faces) is introduced. The possible effects of limited precision on geometric algorithms that operate on such descriptions are investigated. Methods are given for creating and manipulating such descriptions that produce topologically consistent object definitions. The methods minimize the occurrence of ambiguities and indicate those situations in which they may occur. These techniques have been incorporated into a program that computes constructive solid geometry (CSG) operations on polyhedral boundaries.

*This research was supported by the Semiconductor Research Corporation and Tektronix, Inc.

1 Introduction

The description of a geometric object in n-dimensions can be thought of as specifying two types of information: topological data, specifying the connectivities among various object features, and metric data, specifying the positions in space of those features. For instance, a three-dimensional polyhedral boundary is sometimes given by a data structure encoding the topology of the of vertices, edges, and faces, and a set of coordinates encoding their positions. On the other hand, a half-space representation of the same object may contain no explicit topological information but only metric data specifying the various bounding planes.

Converting a half-space representation to a boundary representation requires computing topological data from metric data. Other algorithms that require this kind of computation include: determining if two lines intersect in two dimensions, finding the convex hull of a set of vertices in n dimensions, and boundary-based constructive solid geometry (CSG) operations[1]. The required computations appear straightforward: intersections among the various features must be found.

The difficulty arises when a topological determination must be made from *inexact* metric data. The metric data may be inexact because the process that creates a geometric object may not supply exact coordinates. The locations of an object's features are uncertain if it is scanned-in or digitized. In addition, roundoff errors may accumulate during computations that create the object. The resulting uncertainties about the precise values of coordinates can lead to topological inconsistency. For instance, if a

vertex's coordinates are uncertain, how can one determine if it lies in the plane of a particular face?

There are two issues in this context. First, an object's topology should be well-defined by its representation. Second, several well-defined objects may be presented to an algorithm that is to produce a new object (i.e. the boundary of an object represented as a CSG tree) with well-defined topology.

Several methods have been proposed to address these issues. One method, exact rational arithmetic, attempts to represent all metric data exactly as rational numbers[2], because intersections among features with rational coordinates have rational coordinates. The difficulty is that some geometric data cannot be represented exactly with rational coordinates; distances (which require computation of a square root) and values of trigonometric functions (which appear in rotation matrices) are two examples. This difficulty can be eliminated if one allows an arbitrary symbolic formula for each metric datum[3][4]. However, at some point the formula must be evaluated. If the coefficients and arguments are not given exactly, topological data derived from such a result may still be ambiguous.

Another approach is to assume the input metric data to be exact[5]. Given sufficient precision, a calculation based on exact data can achieve any accuracy. The difficulty with this approach is that the output is still of limited accuracy, so the results from one calculation cannot be used as input to the next. One possible fix is to place a grid over the computed object and snap features to it[6]. The problem is that the snapping process

may be forced to incorporate large portions of the grid into the object, creating many small "stair case" features.

Another method is to overcome roundoff error during algorithm operation by backing up and increasing the precision of an earlier calculation if it is later found that its accuracy is not great enough to resolve a topological determination[7]. Instead of recomputing previous calculations, their results may instead be perturbed as necessary so that they do not introduce ambiguity[8][9]. While these methods have been found to reduce the occurrence of topological inconsistencies, they do not eliminate them entirely.

Our approach concedes that numerical errors may accumulate during computation. We give methods, based on interval arithmetic[10], to measure that accumulation. These methods allow one to locate regions in which metric uncertainty may lead to topological ambiguity. Heuristics can be applied in these regions that remove ambiguity if possible and warn the user otherwise.

2 Definitions

This paper refers mainly to geometric objects in two or three dimensions; however, most concepts are expressed without reference to dimension, making an extension to higher dimensions straightforward. For simplicity we discuss only polygonal and polyhedral objects, but we believe that the approach we present can also be applied to curved surfaces.

2.1 Geometry and Topology

There are three *topological features* of concern in three dimensions: vertices, edges, and faces. A vertex is simply a point. An edge is a subset of a line bounded by vertices. A face is a subset of a plane bounded by a non-self-intersecting polygonal curve made up of edges lying in that plane. The subsets need not be connected. Each feature type can be distinguished by its dimension: a vertex has dimension 0, an edge dimension 1, and a face dimension 2. The affine subset of three-space corresponding to each feature (a point for a vertex, a line for an edge, and a plane for a face) is called a *flat* of dimension 0, 1 or 2.

A particular feature is specified by giving *coordinates* that specify its position with respect to a fixed origin and basis. For instance, a vertex is usually specified with three coordinates, while an edge's position might be specified by providing the coordinates of one endpoint and the coordinates of a vector giving its direction and length. In general, there are many ways of providing coordinates that precisely specify a particular topological feature.

A *geometric object* consists of an arbitrary set of particular topological features described in two parts. The first part is a set of coordinates locating each feature in space. This coordinate information is collectively termed *metric* data. The other part is a list of *connectivity relations*. Each connectivity relation comprises a set of geometric features that intersect (called the connectivity *set*), together with a distinguishing feature that represents the intersection. For instance, several edges may meet at a vertex, several

faces may intersect in an edge, or several vertices may coincide in one vertex. We assume that these connectivity relations are structured so as to permit obtaining the set of features impinging on a particular feature, or, given a feature, finding the features it impinges on[11][12].

An object's representation is *consistent* if every intersection among an arbitrary set of features computed from the metric data is represented explicitly in connectivity relations, and, conversely, every connectivity relation represents an intersection computable from the metric information. Under this definition a half-space representation of a bounded object is not consistent because the existence of edges and vertices is not given explicitly.

A *topological property* of a geometric object is a datum that remains invariant under diffeomorphisms of the coordinate space from which the metric information is drawn[13]. A diffeomorphism is an everywhere C^∞ (derivatives of all orders are continuous) invertible function whose inverse is also everywhere C^∞. Connectivity relations remain fixed under such deformations, so they represent topological properties.

Conversely, many topological properties can be derived directly from the connectivity relations (Euler characteristic or genus of a solid, for instance). In fact, the only topological properties that require metric data are those relying on *orientation* (or ordering) information. For instance, determining whether a point lies inside or outside a certain polygon requires a computation involving both connectivity relations (to discover how the polygon's edges are connected) and metric information (to obtain the point's position relative to the edges). The connectivity relations are also used to discover

if the point being tested actually lies on the polygon itself.

Consistency is desirable because the connectivity relations encode most of an object's topology. Even more important, an algorithm that refers to connectivity relations must not have to consult the metric data to ensure the relations are valid. Such reliance on metric data renders the connectivity relations useless and greatly increases the computational expense of any topological determination.

2.2 Numerical Uncertainty

In practice, consistency may be difficult to achieve because an object's metric data is represented with limited precision quantities. Computing an intersection from these inexact quantities (even if the computation is carried out to arbitrary precision) may lead to ambiguous results.

To account for inaccuracy, and to quantify its effects during geometric computation, we introduce a *tolerance*[14] t_i for each topological feature f_i describing the positional uncertainty of its corresponding flat a_i. It is useful to picture a tolerance as defining a *tolerance region* about each flat. The region is composed of those points that lie within a distance of t from the position of the flat given by its (inexact) coordinates (Figure 1).

To construct the tolerance region for a topological feature of dimension n, consider its two parts: the corresponding flat and the set of features of dimension less than n that form the boundary in that flat. (A vertex has no lower dimensional boundary, so its tolerance region is the same as that of a point.)

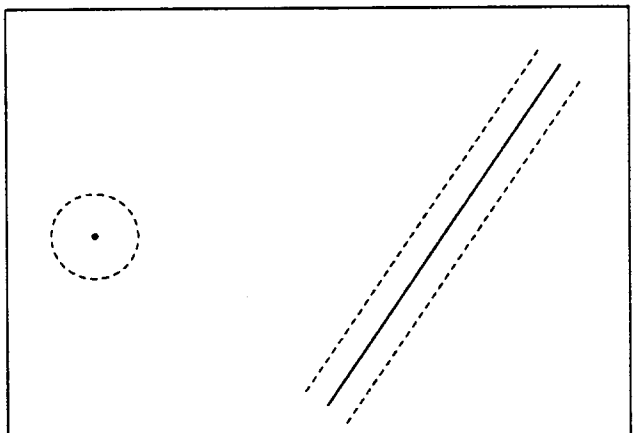


Figure 1. A point and a line and their tolerance regions.

The boundary features as specified by their coordinates, when projected onto the flat, bound a region in that flat (as determined by an orientation on the boundary). The union of this region with the projection of the tolerance regions of the boundary features onto the flat is called the *ideal cut-out region.* The *full cut-out region* is obtained by sweeping the ideal cut-out region in all directions perpendicular to the ideal flat for a distance equal to the flat's tolerance. The tolerance region of a connected component of a feature is the union of the full cut-out region and the tolerance regions of each of the boundary features. The tolerance region of a topological feature consisting of several disjoint parts is the union of the tolerance regions of the connected components.

For instance, the full cut-out region of a connected component of an edge in three dimensions is a cylinder centered on the ideal line bounded

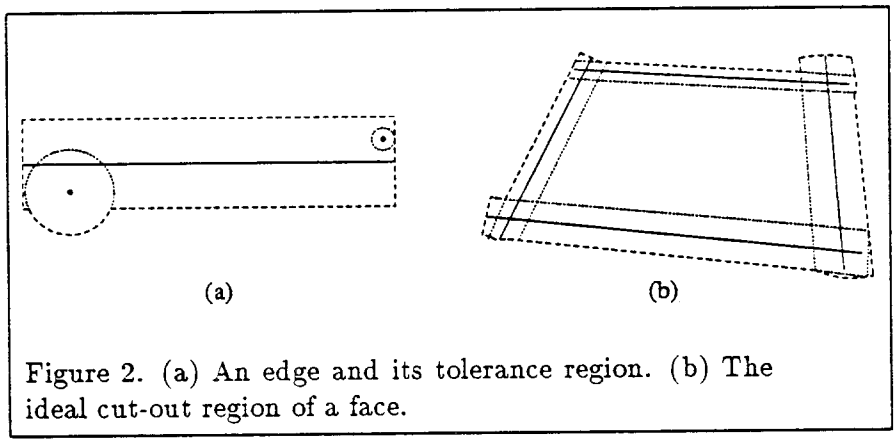


Figure 2. (a) An edge and its tolerance region. (b) The ideal cut-out region of a face.

by planes perpendicular to the ideal line. Each plane is tangent to the tolerance region of the corresponding bounding vertex. The edge's tolerance region is the union of this bounded cylinder and the tolerance regions of the boundary vertices (Figure 2a). Similarly, the tolerance region of a simple n-sided polygon comprises a prismatic slab, n cylinders, and n spheres (Figure 2b).

This method of constructing a feature's tolerance region is not the only one possible. Our definition is simple and leads to straightforward algorithms for determining if features intersect. It also minimizes the amount of tolerance growth required by a geometric algorithm to ensure consistency.

2.3 Intersection

Consider the distance d between two flats a_1 and a_2

$$d = \min\{\|x_1 - x_2\|_2 \mid x_1 \in a_1, x_2 \in a_2\}$$

where $x_i \in a_i$ means x_i is a point of flat a_i. Two flats *intersect within* ϵ if the distance between them does not exceed ϵ. Two flats f_i and f_j *approximately intersect* if they intersect within $\epsilon = t_i + t_j$. That is, if the tolerance regions of two flats intersect, then the flats approximately intersect. Similarly, *approximate intersection of two features* is defined as intersection of the corresponding tolerance regions. Consistency for inexact objects is the same as consistency for exact objects with intersection replaced by approximate intersection.

2.4 Coincidence, Containment, and Alignment

In the exact case, a feature of lower dimension may be *contained* in a feature or flat of higher dimension. Two features or flats of the same dimension may *coincide* if every point of one is a member of the other. We say that a feature f_1 aligns with feature f_2 if f_1 is contained in the f_2's corresponding flat. Any feature contained in or coincident with another is also aligned with it.

A feature f_1 *approximately aligns* with f_2 if every point of f_1's ideal cut-out region is within a distance of $t_1 + t_2$ of the f_2's flat. One feature is *approximately contained* in another if it is approximately aligned with the other feature and if the projection of its ideal cut-out region onto the other feature's ideal flat lies entirely within the other feature's ideal cut-out region. Two features are *approximately coincident* if every boundary feature of one feature approximately coincides with the corresponding boundary of the other. Two vertices are approximately coincident if they approximately

intersect.

2.5 Restrictions on Feature Alignments

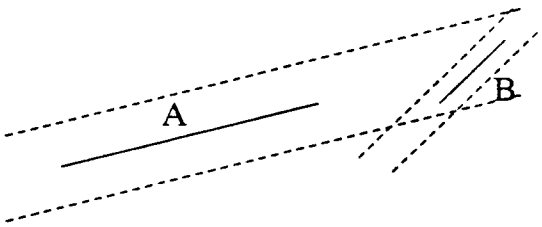
The use of inexact metric data requires some restrictions on the allowable positions of object features if an object is to make sense under the familiar rules of (exact) geometry. These restrictions exclude certain feature constellations whose topology would be unclear. Later sections describe how to convert such a constellation into an allowable constellation.

Consider alignment of features of equal dimension in the exact case. Such alignment is symmetric and transitive: if a_1, a_2, and a_3 are all features of the same dimension, and if a_1 aligns with a_2 then a_2 aligns with a_1; further, if a_2 aligns with a_3, then a_1 aligns with a_3. This is not automatically true for approximate alignment as Figure 3 shows.

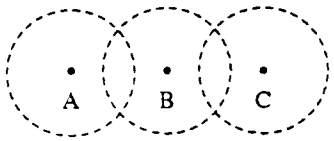
However, for correspondence with the exact case, we also demand symmetry and transitivity of alignment of equal-dimensional features in the approximate case. Furthermore, to simplify data structures, we require that equal-dimensional features never approximately align and intersect simultaneously; such alignment is represented with a single feature.

Similarly, two features may approximately intersect only if their full cut-out regions also intersect. Further, we declare illegal any intersecting feature tolerance regions if the associated boundary tolerance regions intersect in more than one connected component. Figure 4 shows an example of the situation that this restriction is designed to exclude.

Finally, if two distinct features are approximately aligned with and in-



(a)



(b)

Figure 3. (a) non-symmetry of approximate alignment. B aligns with A, but not vice versa. (b) non-transitivity of approximate coincidence. A & B coincide, as do B & C, but A & C do not.

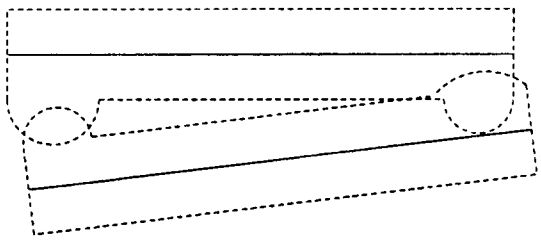


Figure 4. An illegal constellation of edges.

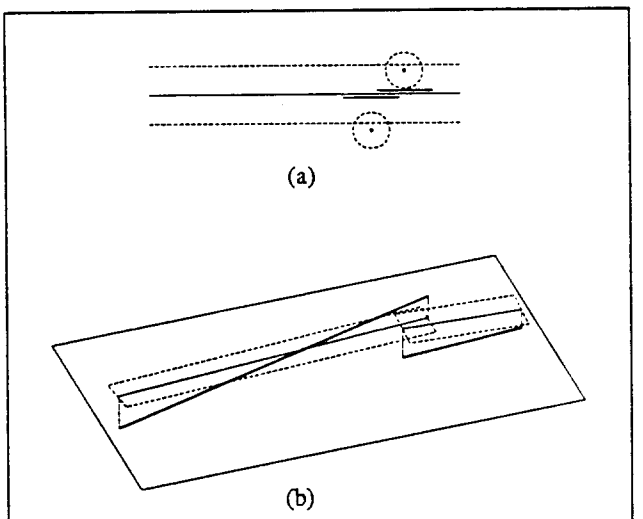


Figure 5. (a) An illegal constellation of vertices about an edge. (b) An illegal constellation of edges about a face.

tersect some higher dimensional feature, and if their projections onto the higher dimensional feature's ideal flat are approximately aligned (Figure 5), then we require the two features to be approximately aligned.

3 Deformations and Perturbations

Suppose one is presented with an object that is consistent for certain specified tolerance values. If this object is subjected to perturbations, such as those that result from the floating point computation of an affine transformation used to place the object in space, it may no longer be consistent for any tolerance values. Even the smallest perturbations may ruin approxi-

mate consistency because a pair of features in the original object may only be separated by a distance slightly larger than the associated tolerances, so that after a perturbation the features' tolerance regions may intersect. Similarly, two features may start off intersecting within ϵ, but after a perturbation may no longer do so. Therefore, the original object must satisfy further requirements if it is to retain its consistency under reasonable perturbations of its metric data.

To avoid these effects, an object's features that do not intersect within ϵ must be separated by a minimum distance that is large relative to ϵ. The minimum distance separating features that do not intersect (as determined by the connectivity relations) is called the *minimum feature separation*[6][15] or μ. Suppose ϵ_0 is the largest tolerance among those associated with the features in some zone of an object and μ is the minimum feature separation in that zone. Such an object can withstand perturbations of the coordinates that locate its features in space as long as those changes do not alter the distances between non-intersecting features by as much as $\frac{\mu}{2} - \epsilon_0$.

There is an even stronger reason for maintaining a minimum feature separation. Most geometric objects locate some of their features implicitly by referring to other features. For instance, an edge is typically given by the coordinates of vertices at its endpoints. If the tolerance is ϵ and the vertices are separated by only a small distance comparable to ϵ, then the edge's direction is poorly determined. This example shows that the μ in a zone should normally be several orders of magnitude greater than the tolerances of the features in that zone.

4 Geometric Algorithms

In the previous sections we have given rules for ensuring that individual geometric objects are topologically consistent and discussed the possibility of maintaining consistency under geometric operations that are not intended to modify topology. However, many geometric algorithms modify the topology of objects presented to them or use these objects to define new objects with new topology.

To generate the connectivity relations, an algorithm must classify groups of two or more features as intersecting or non-intersecting. The metric data that must be examined are the distances between pairs of features. If such a distance is large (on the order of the zone's minimum feature separation), the two features do not intersect. However, if the distance between features is small (about as big as nearby tolerances), the determination is not so clear because discrete topological determinations must be based on inexact continuous metric data.

The determination is arbitrary only if a small change in the relative positions of the two features changes intersection to non-intersection (Figure 6). Disregard for the consequences of this arbitrary decision may cause an algorithm to produce an object with unexpected or undesirable topology.

A solitary arbitrary coincidence determination poses no difficulty. The difficulty comes when one determination is a part of a series. If an algorithm is building a complex object from a set of vertices some of which may be coincident (as may happen in a convex hull algorithm, for instance), one determination may affect the outcome of subsequent ones.

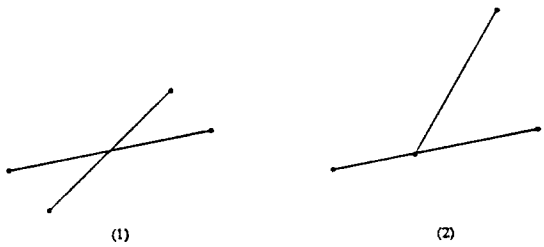


Figure 6. Two edge-edge intersections in R^2. The first is not sensitive to perturbations, but in the second, the vertex of one edge is coincident with the other edge, so a small change in the position of either edge can make the edges disjoint.

At any given time, an algorithm may make a topological determination based on metric data and on previous topological determinations encoded in the emerging connectivity relations. An ideal topological determination is independent of exactly which determinations led up to it; in particular, it is independent of the order in which the determinations are made. However, a practical *numerical* topological determination may not possess this property. We say that a numerical topological determination is *ambiguous* if the particular set of previous determinations affects its outcome. Otherwise it is *unambiguous*.

A serious problem stemming from ambiguity involves the transitivity of coincidence. Suppose the tolerance regions of two vertices, A and C, are found to be disjoint, ruling the vertices non-coincident. Then a third vertex, B, is examined, and ruled coincident with both A and C (Figure

3). This implies that A and C are coincident, contradicting the earlier determination.

This example implies that an unambiguous determination cannot rely on previous determinations and may even lead to their revision. It also shows that the union of tolerance regions provides a useful new tolerance region for a group of features found to be coincident. The difficulty is that the shape of the union of tolerance regions can become arbitrarily complicated as more and more coincident features are added. Keeping track of such a region is tantamount to using a symbolic approach[16][17] to represent feature intersections—each feature present in the coincidence adds a term to the formula representing the union of tolerance regions.

We reject such complex approaches because they do not eliminate the need to check whether previous determinations must be reversed as new features are considered. Instead, we recognize that ambiguous determinations can lead to consistent objects as long as the results of such determinations are properly recorded in the connectivity relations. This fact allows us to use a simplified tolerance region, described by a single tolerance, for coincident features. This region has the same basic form as those of the original features.

5 Practical Solutions

A practical algorithm can be robust in spite of having to make ambiguous topological determinations. However, an algorithm must be carefully designed if it is to produce sensible results. A solid modeler that always

produces a single vertex as output is certainly robust, but not very useful.

We assume that the input objects' tolerances are much smaller than the associated minimum feature separations. Similarly, an algorithm's output should satisfy the same criterion. If this is impossible, as may occur if an algorithm is forced to generate a topologically complex region of small size, the algorithm should generate a warning message.

5.1 Tolerance Handling

Our method treats geometric objects consisting of vertices, edges, and faces. A set of coordinates is kept for each feature determining the location in space of the feature's associated flat: a position for a vertex, a parametric line equation for an edge, and an implicit plane equation for a face. In addition, a single tolerance is kept with each feature to quantify the accuracy of its metric data. This tolerance defines a spherical region about a vertex's approximate position, a cylindrical region about an edge's position, and a slab about a face's position.

Generating the connectivity relations from metric data means determining those pairs of features that intersect. In principle this means checking whether their associated tolerance regions intersect.

If a pair of features have been determined to intersect, appropriate modifications must be made to the geometric object. In the case of a non-aligned intersection, a new feature (with an associated tolerance) must be created to represent it. In the case of an alignment, the tolerances of the coinciding features may have to be adjusted. For a coincidence, the pair of features is

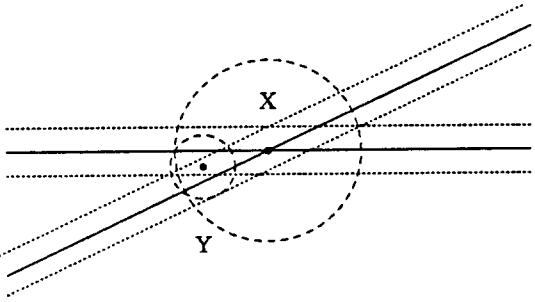


Figure 7. The tolerance region assigned to a computed intersection. Vertex X (the intersection) must be found coincident with vertex Y.

replaced with a single feature with a suitable tolerance.

5.1.1 Intersection

To compute the intersection of a pair of non-aligned features, an accurate computation[5][18] using the approximate coordinates of each member of the pair is employed to find the best approximate coordinates for the intersection feature. The tolerance for the new feature is made large enough so that the corresponding tolerance region encloses the intersection of the two original tolerance regions. Approximate consistency requires a region of this size because if another feature intersects both of the original features' regions, then it impinges on their regions' intersection so that it must also impinge on the computed intersection feature (Figure 7). In addition, if the roundoff error associated with the intersection computation is not

small relative to the computed intersection tolerance, then the intersection tolerance is increased by an appropriate error estimate to account for the inaccuracy.

As an example, consider finding the intersection of an edge with a face. The first step is to check that the edge or any of its boundary vertices are not contained in the face's plane (see below). If the edge itself is not coincident with the face's plane but one of its bounding vertices is, then the edge intersects the face's plane at that vertex. Otherwise, the intersection point of the edge's ideal line with the face's ideal plane is found using an accurate computation (typically accomplished by substituting the line's parametric equation into the plane's parametric equation and solving for the parameter value). If the computed point lies outside the edge (as determined by its bounding vertices and the corresponding tolerance regions) the edge does not intersect the plane. Otherwise, the point is assigned a tolerance large enough so that the corresponding region encloses the intersection of the line's and plane's tolerance regions. Next, a test must be made to see if the point lies in the face's interior or on its boundary. As the point is checked against each edge in the boundary, a test is also made for containment of the point within that edge or coincidence with one of its bounding vertices.

5.1.2 Alignments

We distinguish four ways that a feature f_1 with tolerance t_1 may align *and* intersect with a feature f_2 with tolerance t_2:

1. The ideal cut-out region of f_1 lies within a distance of $t_1 + t_2$ of f_2's ideal flat. This case covers approximate alignment.

2. Neither feature is a vertex, their tolerance regions intersect, and the cosine of the angle between their flats is on the order of $\frac{t_i+t_j}{2\mu}$, where μ is the minimum feature separation of the input. (In higher dimensions, the cosine of the angle is replaced with the condition number of the intersection computation.) This case is designed to detect intersections that, if computed, would require an unacceptably large tolerance region. Exactly how large is unacceptable depends on the application.

3. Each feature approximately intersects and aligns with a higher dimensional feature f_3, and f_1's projection onto f_3's ideal flat approximately aligns with f_2's projection onto it. This case is designed to remedy illegal feature constellations such the ones shown in Figure 5.

4. If a feature of dimension d is approximately contained in a feature of dimension $d+1$, and the $d+1$ dimensional feature is approximately contained in a feature of dimension $d+2$, then the d dimensional feature is also approximately contained in the $d+2$ dimensional feature. In three dimensions, a vertex may be contained in a face by being contained in one of the face's bounding edges.

If a lower dimensional feature aligns with a higher dimensional one, the intersection of the features is found by projecting the lower dimensional feature's tolerance region onto the higher dimensional feature's flat and

carrying out the intersection computation in that flat (see above). Once the intersection is known, an appropriate entry is made in the connectivity relations. This includes some relations that are not always considered; for instance, a vertex impinging on a face's interior creates a degenerate contour in that face.

The tolerance values of the associated features are left unchanged if the alignment results from case (1). Otherwise, the tolerance of the lower dimensional feature is increased so that its full cut-out region intersects the full cut-out region of the higher dimensional feature.

If one feature intersects and aligns with another of the same dimension, they are merged into a single feature. A new flat is created whose coordinates are simply the average of the coordinates of the two aligned features. The two original features partition one another into disjoint regions. The boundaries of these regions are computed by projecting the tolerance regions of both features into the newly created flat (for a pair of vertices, there are no boundaries so there is no partitioning) and finding intersections in that flat. These partitioned boundary features form the boundary features of the new feature. All references to the two features in the connectivity relations are replaced with references to the new feature. The tolerance for the new feature is made large enough so that the corresponding tolerance region encloses the union of the aligned tolerance regions. Figure 8 shows how two collinear edges are converted into a single multi-segment edge.

Consider discovering if a vertex is contained in a particular face. In this example, case (2) cannot arise and we start with a check for case (4). First

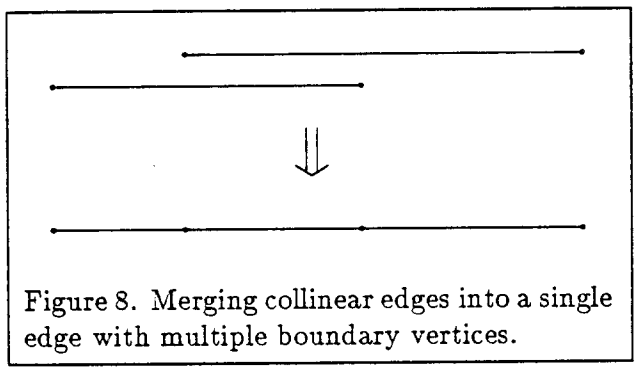


Figure 8. Merging collinear edges into a single edge with multiple boundary vertices.

we check whether the vertex approximately intersects any of the vertices in the face's boundary. Next we check for approximate containment in any of the face's edges. If the vertex is contained in an edge, it must be re-checked for possible coincidence with the edge's bounding vertices under case (3). If none of these tests indicate that the vertex lies on the face's boundary, the vertex's distance from the plane is found by substituting its coordinates into the face's plane equation. If the vertex does not approximately intersect the face's plane, then it does not lie within the face. If it does approximately intersect the plane, it may still coincide with one of the boundary features under case (3). This case is checked for by projecting the boundary feature's (vertices and edges) tolerance regions onto the face's plane and checking if any intersect the projection of the vertex's tolerance region onto the plane.

A vertex that approximately intersects the face's plane but does not lie on a face boundary is contained in the face if it is in the face's interior (case (1)). The test is carried out in the same way as for an intersection point of an edge with a face's plane. A clever algorithm can interleave most of

these tests, requiring only one pass over the face's boundary features.

A vertex found intersecting a boundary feature has its tolerance adjusted so that its tolerance region intersects both that of the feature and that of the face (as required by case (4)). A vertex intersecting the face but not its boundary needs no tolerance adjustment.

5.2 Backtracking

The techniques presented to determine intersections and to adjust tolerances provide a comprehensive method to produce consistent connectivity relations in many cases. However, some constellations of features require more complex techniques to ensure consistency.

Difficulties arise from the growth of the tolerance regions of intersecting or coincident features because previously non-intersecting features may intersect after tolerances have been increased (Figure 9). If no information is kept about the relative separation of features, then anytime an intersection is found, all features previously considered as possibly intersecting either member of the intersecting pair must be re-examined to determine if any of their tolerance regions intersect the new tolerance region. If at least one does, then this other intersection has to be accounted for and the search process repeated, possibly leading to a long chain of re-testing previously considered features. The required checking considerably increases the cost of intersection determination.

The average cost can be radically decreased by keeping a number with each feature representing the smallest distance between it and all other fea-

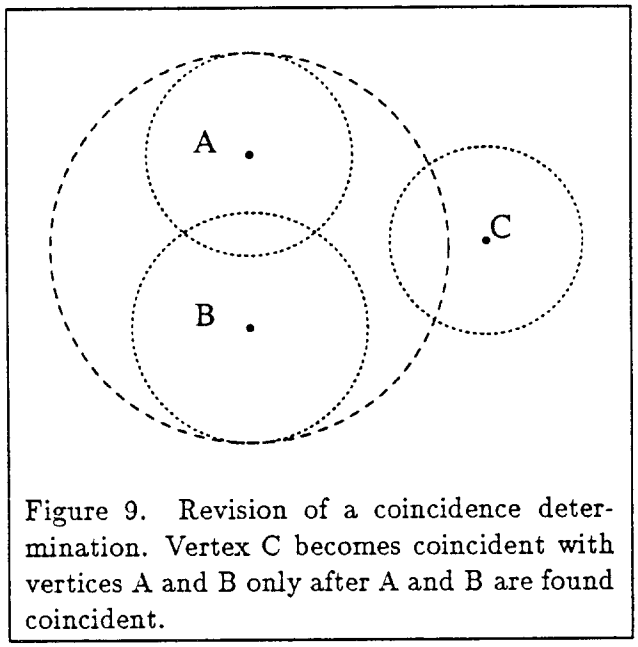


Figure 9. Revision of a coincidence determination. Vertex C becomes coincident with vertices A and B only after A and B are found coincident.

tures that have been determined not to intersect it. When two features are checked for intersection, the minimum distance stored with each feature is compared to the computed distance and updated as appropriate. When two features are found to intersect, the stored minimum distances are compared to the tolerance value for the computed intersection. If the stored distance is larger than the new tolerance, no backtracking is required. Otherwise, the previously examined features are re-examined as already described. If μ of the individual objects is large relative to tolerances, backtracking will be required rarely, if at all.

Another advantage of saving the minimum non-intersecting feature distance is that the μ of the final object can be obtained by taking the mini-

mum over all features of the stored distances at the end of the computation.

6 Implementation

We have implemented a solid modeling algorithm that incorporates these methods[19]. The algorithm accepts two boundary representations of polyhedra and produces the boundary representation of the regularized union, difference, or intersection. Boundaries are vertex based; the locations of edges and faces are derived from vertex coordinates. However, during the algorithm's operation, coordinates are kept for each feature, be it vertex, edge, or face. Each feature also has an associated tolerance. Vertex tolerances are assigned an arbitrary small value when a boundary is read in; an edge tolerance is derived from the tolerances of its bounding vertices. Face tolerances are computed so that each face tolerance region contains all the tolerance regions of the vertices that lie within it. The minimum feature size for the input objects is assumed to be large relative to the tolerances.

The algorithm operates by considering two faces, one from each boundary, at a time. The faces are first checked to see if they are aligned. If they are not, the vertices from one face are considered relative to the other face. If two vertices at either end of an edge lie on opposite sides of the face's plane, a vertex representing the intersection of the edge with the face's plane is computed and a tolerance assigned. After all the edges of one face have been checked against the other, the faces' roles are reversed.

Eventually all the intersections of each face's boundary with the other's plane are located. This includes vertices of one face's boundary that are

found coincident with the other's plane. These points are sorted into order along the computed ideal intersection line, and the information is used to determine the actual intersection spans of the two faces so that appropriate modifications can be made to the face's boundaries. The sorting also allows identification of intersection points that coincide with one of the faces' boundaries. Once coincidence determination and partitioning have been completed for all face pairs, the solid modeling result is obtained by selecting the appropriate portions of the partitioned faces.

The algorithm uses tolerances to generate the connectivity relations that define the partitioned boundaries. However, the current implementation does no backtracking. Instead, it is assumed that backtracking is unnecessary as long as tolerances remain small relative to the minimum feature separation in the input objects. If a computed tolerance exceeds a user settable threshold, the algorithm generates a warning message and continues its operation. Further, upon termination, the algorithm abandons individual sets of coordinates for each feature type and describes output objects in terms of vertex coordinates only.

Even with this simplified approach, the algorithm has been found to be robust. On the rare occasions that the algorithm has produced inconsistent output (because of the lack of backtracking), it has always produced a warning message. For example, if an attempt is made to compute the intersection of two congruent cubes, one rotated by a small angle about some axis relative to the other[20], a correct result is obtained for angles greater than about 0.01°. For angles less than about 0.0005° the reasonable

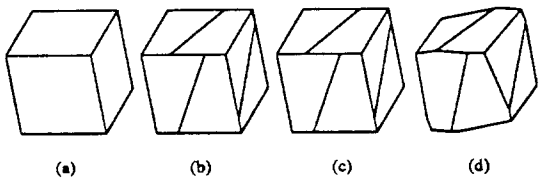


Figure 10. Output of solid modeling algorithm given two congruent cubes, one rotated slightly from the other. In (a), the angle is 0.0001° and the algorithm produces a single cube as result. In (c), the angle is 0.05°. In (b), the angle is 0.001°; with the nominal tolerances the algorithm reported errors and failed to produce the boundary of a solid as output. The tolerances were decreased to produce the scene pictured here. For comparison, (d) shows the result with an angle of 15°.

result of a single cube is produced. For angles between 0.0005° and 0.01° an incorrect result is sometimes created, but not before warnings have been produced. After appropriately adjusting the default tolerance values, the algorithm can be rerun to produce an acceptable result (with no warnings reported) for angles within the given range (Figure 10).

Although backtracking would eliminate any inconsistency, large tolerances are still a reason for concern because they imply that the location of a computed feature is poorly defined. Whenever an algorithm produces a large tolerance for a computed feature, a warning is appropriate to alert the user to possible inaccuracies in the computed object. Large inaccuracy may indicate unexpected topology, such as the reduction of a complex region to

a single vertex.

A serious drawback of the algorithm as implemented is that there is no provision to discover the minimum feature separation of the output boundary. If the algorithm were to become a part of a solid modeling tool in which the output of one modeling operation were used as the input to another, this information would be essential. It could also ensure that outputting metric data in terms of vertex coordinates only does not introduce perturbations large enough to jeopardize consistency.

7 Conclusion

A method for constructing geometric representations that are consistent in the face of moderate numerical inaccuracy has been presented. Suitable rules governing the use of these representations can lead to robustness in geometric algorithms. Complete robustness (in the sense of an algorithm producing a geometric result that adheres to the same constraints as its inputs) may only be possible at the expense of simplification of and undesirable alterations to small, topologically complex regions. However, the use of tolerances precisely quantifies the effects of numerical inaccuracy on geometric representations and provides a basis for reliable methods to extract topological data where this is possible without ambiguity. Our method detects ambiguity and identifies situations in which the numerical data could lead to unexpected topological changes.

The tolerance model could be extended in several ways to reduce the number of instances of zones that are determined to be topologically am-

biguous. One idea is to use a relaxation method in an attempt to reduce tolerances of geometric objects. Smaller tolerances in individual objects would decrease the likelihood of an algorithm creating an unacceptably large tolerance region when several objects are combined. Features' coordinates could be perturbed as to reduce an objective function made up of a weighted sum of all the features' tolerance values. The reason that relaxation is a promising method to achieve tolerance reduction is that the perturbations required to reduce a single feature's tolerance are restricted to the coordinates of nearby features.

However, roundoff errors do accumulate during consecutive computations, and there are cases in which tolerances cannot be reduced. Figure 11 shows an object whose boundary is made of quadrilateral faces. These faces are non-planar—the twisted nature of the object does not admit planar quadrilateral faces. No amount of local perturbation of vertex positions can alter this fact.

Tolerance methods can in principle be extended to cover topological determinations on curved objects. The principles are essentially the same: a curve or surface is endowed with a tolerance that defines a small region about it, and this region is used to test for intersection with other features. How the shape of this region must vary over the extent of the curve or surface must be considered before an appropriate tolerance model can be developed. For instance, computation of the intersection curve of two surfaces is typically carried out numerically, so the tolerance of the intersection curve depends not only on the tolerances of the intersecting surfaces, but

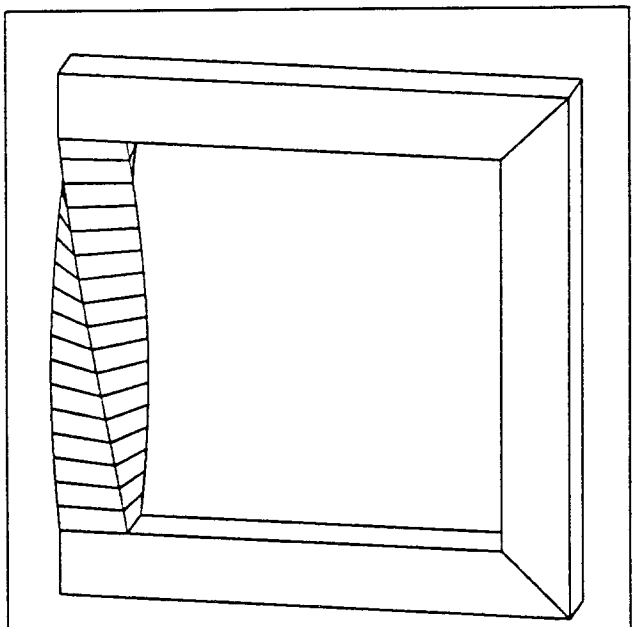


Figure 11. An inexact object whose tolerances cannot be reduced. μ is about 0.5, while the maximum face tolerance (with vertex tolerances of zero) is about 0.033; the ratio of μ to tolerances is thus only about 15.

possibly on the accuracy of the algorithm used to compute the piecewise linear approximation to the intersection curve as well.

References

[1] Martti Mäntylä. Boolean operations of 2-manifolds through vertex neighborhood classification. *ACM Transactions on Graphics*, 5(1):1–29, January 1986.

[2] W. Randolph Franklin, Peter Y.F. Wu, and Sumitro Samaddar. Prolog and geometry projects. *IEEE CG & A*, 6(11):46–55, November 1986.

[3] John Francis Canny. *The Complexity of Robot Motion Planning.* PhD thesis, MIT, 1987.

[4] Dennis S. Arnon. Topologically reliable display of algebraic curves. In *Proceedings of SIGGRAPH*, pages 219–227, 1983.

[5] Thomas Ottmann, Gerald Thiemt, and Christian Ullrich. Numerical stability of geometric algorithms. In *Proceedings of the Third ACM Symposium on Computational Geometry*, pages 119–125, Waterloo, Ontario, 1987.

[6] Mark Segal and Carlo H. Séquin. Consistent calculations for solid modeling. In *Proceedings of the First ACM Symposium on Computational Geometry*, pages 29–38, 1985.

[7] David Dobkin and Deborah Silver. Recipes for geometry & numerical analysis—part I: an empirical study. In *Proceedings of the Fourth ACM Symposium on Computational Geometry*, pages 93–105, Urbana, Illinois, 1988.

[8] Cristoph M. Hoffmann, John E. Hopcroft, and Michael S. Karasick. *Robust Set Operations on Polyhedral Solids.* Technical Report 87-875, Computer Science Dept., Cornell University, 1987.

[9] Cristoph M. Hoffmann, John E. Hopcroft, and Michael S. Karasick. Towards implementing robust geometric computations. In *Proceedings of the Fourth ACM Symposium on Computational Geometry*, pages 106–117, Urbana, Illinois, 1988.

[10] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis.* Springer-Verlag, New York, 1980.

[11] B. G. Baumgart. *Geometric Modelling for Computer Vision.* Technical Report STAN-CS-74-463, Stanford AI Lab, 1974.

[12] Kevin Weiler. Edge-based data structures for solid modeling in curved-surface environments. *IEEE CG & A*, 5(1):21–40, January 1985.

[13] V. Guilleman and A. Pollack. *Differential Topology.* Prentice-Hall, Englewood Cliffs, NJ, 1974.

[14] Aristides A. G. Requicha. Toward a theory of geometric tolerancing. *International Journal of Robotics Research*, 2(4):45–60, Winter 1983.

[15] Victor Milenkovic. Verifiable implementations of geometric algorithms using finite precision arithmetic. In *International Workshop on Geometric Reasoning*, Oxford, England, July 1986.

[16] Herbert Edelsbrunner and Ernst Peter Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. In *Proceedings of the Fourth ACM Symposium on Computational Geometry*, pages 118–133, Urbana-Champaign, 1988.

[17] Chee-Keng Yap. A geometric consistency theorem for a symbolic perturbation scheme. In *Proceedings of the Fourth ACM Symposium on Computational Geometry*, pages 134–142, Urbana, Illinois, 1988.

[18] G.H. Golub and C.F. Van Loan. *Matrix Computations.* Johns Hopkins Press, Baltimore, MD, 1983.

[19] Mark Segal and Carlo H. Séquin. Partitioning polyhedral objects into non-intersecting parts. *IEEE CG & A*, 8(1):53–67, January 1988.

[20] David H. Laidlaw, W. Benjamin Trumbore, and John F. Hughes. Constructive solid geometry for polyhedral objects. In *Proceedings of SIGGRAPH*, pages 161–170, 1986.