# Robust Set Operations on Polyhedral Solids

Christoph M. Hoffmann[1]
John E. Hopcroft[2]
Michael S. Karasick[2 3]

87-875
October 1987

Department of Computer Science
Cornell University
Ithaca, New York 14853-7501

[1]Supported in part by the Office of Naval Research under contract N00014-86-K-0465
[2]Supported in part by the Office of Naval Research under contract N00014-86-K-0281 and the National Science Foundation under grant DMC 86-17335 and grant DCR 85-02568.
[3]Supported in part by a National Science and Engineering Research Council (Canada) Postgraduate Fellowship.

# Robust Set Operations on Polyhedral Solids

Christoph M. Hoffmann[1]
Computer Science Department
Purdue University

John E. Hopcroft[2]
Computer Science Department
Cornell University

Michael S. Karasick[2 3]
Computer Science Department
Cornell University
and
Computer Science Department
McGill University

**Abstract**

We describe an algorithm for performing regularized set operations on polyhedral solids. Robustness of this algorithm is achieved by adding symbolic reasoning as a supplemental step that compensates for possible numerical uncertainty. The algorithm has been implemented, and our experience with the implementation is discussed.

---

[1] Supported in part by the Office for Naval Research under contract N00014-86-K-0465.

[2] Supported in part by the Office of Naval Research under contract N00014-86-K-0281 and the National Science Foundation under grant DMC 86-17335 and grant DCR 85-02568.

[3] Supported in part by a National Science and Engineering Research Council (Canada) Postgraduate Fellowship

## 1. Introduction

We present a robust algorithm for performing regularized set operations on polyhedral solids described using a boundary representation. That is, we give a reliable method for the regularized intersection, union, difference and complement of polyhedral solids.

Algorithms for regularized set operations on polyhedral objects have been implemented before [Braid, 1975; Wesley, Lozano-Perez, Lieberman, Lavin, and Grossman, 1980; Requicha and Voelcker, 1985; Laidlaw and Hughes, 1986; Mantyla, 1986; Owen, 1986]. However, the robustness problem has not been addressed deeply, and certain input configurations of simple objects may lead to failure. An example may illustrate the problem: Consider a unit cube. Take a second cube, obtained from the first by successive rotation about each principal axis by a small angle, and intersect it with the first cube. Many polyhedral modelers fail when the angle of rotation drops below two degrees because these two cubes are sufficiently similar that errors are made when computing this intersection [Laidlaw and Hughes, 1986]. In contrast, our algorithm does not fail at any angle.

Simply put, the robustness problem is rooted in floating point arithmetic. While floating point calculation can distinguish object features that are sufficiently separated, it can never reliably determine their coincidence. Moreover, in a certain region of proximity, floating point computation will give seemingly random results due to round-off errors. This region of criticality, in which many modelers fail, depends on the machine precision and on the nature of the computation. It cannot be addressed satisfactorily by declaring two features coincident whenever they are closer than some tolerance $\varepsilon$. Doing so leads to inconsistent decisions in certain situations. A more sophisticated approach is needed.

Because incidence testing is a fundamental operation in an intersection algorithm, one is ill-advised to base these tests on floating point calculations alone. Conversely, using purely symbolic calculation or exact arithmetic is not the answer because of inefficiency and the fact that the original data is often inexact: For example, four planes that are meant to intersect at a common vertex probably intersect in sets of three at four distinct points near the vertex. An approach is needed that satisfies the following criteria:

(1) the method must be efficient;

(2) it must account for data imprecision; and

(3) it must make consistent incidence decisions.

In this paper, we give a preliminary formulation of such a method that relies on floating point calculation when this is safe, and deduces relative position symbolically and reproducibly when floating point calculation yields ambiguous results. It is perhaps not possible to formulate a complete and consistent general calculus for all geometric computations while maintaining the simplicity and efficiency of our approach, but it appears to be possible for each specific geometric operation, and in particular, for regularized set operations on polyhedral solids.

Our paper has two major parts: The incidence tests, which are of crucial importance for achieving robustness in our algorithm, are outlined in Section 3. Subsequent sections describe the algorithm itself. Two aspects make it worthwhile giving this algorithm in detail. Since incidence tests are not necessarily symmetric, their asymmetry must be reflected in the design of our algorithm. Moreover, as noted in [Laidlaw and Hughes, 1986], most published accounts of polyhedral modelers leave out crucial aspects needed to implement them completely. Complete descriptions of our incidence tests and regularized intersection algorithm are found in [Karasick, 1988].

## 2. Experience with the Algorithm

Our algorithm has been implemented in Common Lisp on a Symbolics Lisp machine. Predecessors of the algorithm were implemented in Interlisp on a Xerox Dandelion and provided experience with alternative solid representations. Extensive experimentation convinced us that the non-manifold representation, described below in Section 4, yields the simplest algorithm. One of the difficulties encountered when representing non-manifolds using manifold representations [Weiler, 1984], is that coincidences between two solids discovered when intersecting them can be interpreted topologically in different ways. Some of these interpretations lead to faces of zero area, and the resulting structures demand very intricate techniques. This is because the topology at non-manifold points cannot be explicitly described using a manifold representation. The implementation described in [Paoluzzi, Ramella and Santarelli, 1986] succeeds in choosing a topology that avoids such degeneracies but requires triangulating faces, a severe constraint if one would like to extend the modeler later to the curved surface domain. In contrast, virtually all difficulties connected to special object positions disappear when using a non-manifold representation, and full attention can be given to robustness issues.

As a simple test object for robustness we used the unit-cube example of Section 1. After rotating a unit cube about each principal axis by a small amount, we obtained a second cube with which the first one was intersected. As stated in [Laidlaw and Hughes, 1986], most existing modelers break when the angle of rotation is less than 2 degrees. With the heuristics advocated by [Laidlaw and Hughes, 1986], their modeler breaks at about 1/2 degree. We were unable to break our modeler after we added to it the incidence rules described in Section 3. Instead, for rotations of up to about $1/200^{th}$ of a degree, the result is the intersection, an object with 16 vertices. Below about $1/200^{th}$ of a degree the algorithm concludes that two cubes are identical and returns one of the cubes as the intersection.

A nonregular heptahedron, shown in Figure 2.1, was rotated through various angles and intersected with itself. As the angle of rotation was decreased, the intersection gracefully converged to the original heptahedron as various vertex-pairs were deemed coincident. Final convergence occurred at $1/10{,}000^{th}$ of a degree.

A polyhedral approximation to a sphere was obtained by starting with a cube, and using the following iterative procedure: rotate the original cube 45 degrees about the x-axis, and intersect the rotated cube with the original cube; next, rotate this result 45 degrees around the y-axis, and intersect; rotate this result 45 degrees around the z-axis, and intersect; and repeat this cycle until the object has 1000 faces. The final object, shown in Figure 2.2, has 3034 faces, 8236 edges and 5204 vertices, and was generated at the end of three cycles (twelve intersections). The time used by the intersection algorithm to compute each intersection is shown below. Although the problem of intersecting solids has a quadratic lower bound, performance close to linear is obtained on examples such as this by using a preprocesser (see Section 5.1) to eliminate extraneous computation.

| Time to generate (sec) | | Output size | | |
|---|---|---|---|---|
| *Preprocessing* | *Total intersection* | *Faces* | *Edges* | *Vertices* |
| | | 6 | 12 | 8 (cube) |
| 0 | 5 | 10 | 24 | 16 |
| 0 | 10 | 18 | 36 | 20 |
| 2 | 21 | 34 | 84 | 52 |
| 2 | 55 | 58 | 144 | 88 |
| 5 | 71 | 98 | 226 | 130 |
| 14 | 128 | 162 | 406 | 246 |
| 28 | 242 | 266 | 676 | 412 |
| 63 | 434 | 434 | 1076 | 644 |
| 129 | 771 | 710 | 1778 | 1070 |
| 268 | 1536 | 1150 | 2988 | 1840 |
| 501 | 3306 | 1866 | 4880 | 3016 |
| 1085 | 6751 | 3034 | 8236 | 5204 |

The principal limitation of the incidence tests as implemented is that they were designed based on valence-three vertices. Thus if two solids coincide at a vertex with high valence and floating point computations demonstrate that symbolic-inference tests are necessary to describe this coincidence, then our intersection algorithm may fail. More research is needed to derive a consistent and complete set of incidence tests for geometric computations in general and intersection algorithms in particular.

## 3. Robustness in the Operations

Robustness is achieved primarily by designing reliable basic operations with which the algorithm is implemented. Our approach is based on two concepts:

- By understanding the inherent error of the floating-point calculations involved, we can distinguish between trustworthy and inconclusive results.
- Since the input data is not exact, we must reduce its numerical redundancy and limit the size of surface features not to be below a given tolerance.

In this section we elaborate on the first concept.

The basic operations and tests on which the consistency and correctness of the algorithm depend are demonstrated by the following examples:

**Incidence tests:** Does a vertex lie on a plane, do two vertices coincide, or do two edges intersect?

**Ordering operations:** What is the relative order of points along a line, and what is the radial order of directed lines originating in a common point?

**Pairing operations:** Which pair of faces enclose volume at an edge, and which pair of edges enclose face area at a vertex?

What is fundamentally different about incidence tests (as opposed to the ordering and pairing operations) is that we are comparing features from distinct objects and thus these features can be arbitrarily close: they can be closer than the accuracy of the model data. Thus, there is no guarantee that all ambiguities can be resolved, and we will have to make an arbitrary choice at some point. It is important to insure that this choice is consistent with related choices.

Whenever a decision is needed as to whether a feature of one object coincides with a feature of another object, we perform the necessary numerical computation. If the features are separated by some predetermined tolerance we can safely assume that the features do not coincide. However, we can never numerically determine that features do coincide due to the uncertainty caused by numerical round off. Thus, we need some way to make a positive decision when features are within this tolerance. Whenever features are sufficiently close, we are free to conclude that they do or do not coincide, provided that we do not make a decision that is inconsistent with some known fact or previous decision.

At first it appears that a powerful theorem prover is necessary to determine whether a given decision is independent of previously made decisions. However, by understanding the types of inconsistencies that arise in a limited domain (such as set operations on solids), we can develop a small set of tests that maintain consistency for situations that arise in a specific algorithm. Whenever we must make a logical decision about the relative position of two possibly coincident features, we apply these tests to see if they separate these features. If not, then we say that these features coincide. The reason for saying this is that our experience in such situations indicates that features coincide and we are simply observing the result of numerical round off. At some small separation we must declare that two features do coincide or we will get many features topologically distinct, but infinitesimally separated. (We will still get a certain number of small structures, but they will be necessary to keep the topology consistent during construction of the resulting object.) Afterwards a post processor might be used to perturb the face equations so as to eliminate structures of size smaller than some tolerance. In this way a new object is constructed that has some minimum feature separation.

This philosophy of using symbolic reasoning to insure consistency of logical decisions when using numeric calculations should have wide-spread application in areas outside geometric modeling.

### 3.1. Accuracy

The accuracy and dependability of a floating point calculation depends on both the machine precision and on the computation at hand. With each numerical computation is associated an uncertainty estimate, $\varepsilon$. A logical decision based on two numerical values is reliable, provided the values differ by at least the sum of the two uncertainty estimates. If the numerical computation does not support a reliable logical decision, then additional symbolic computation is done to resolve the ambiguity.

Two basic floating point calculations are done for testing incidence:

(1) Given a vertex defined by the intersection of three planes, compute its coordinates.

(2) Given point $a$ and plane equation $P$, evaluate $P(a)$ in order to determine whether $a$ is on $P$.

In each case, standard estimates exist for deriving bounds on the precision of the result [Forsythe and Moler, 1967], and these estimates are used to judge whether the result is conclusive.

While it is not possible to exceed machine precision efficiently, one can easily assume less precision than is actually delivered. We are able to do this by supplying an input parameter to the algorithm specifying a nominal machine precision. (This is very useful when studying experimentally the effects of changing the tests for deciding incidence.)

Let $\varepsilon_a$ denote the error for determining the coordinates of point $a$. Points $a$ and $b$ are called *distant* if they are separated by at least $\varepsilon_a + \varepsilon_b$. Otherwise, these two points are *near* and they may or may not coincide. Now assume that we test whether $a$ lies on plane $P$. The accuracy of this answer, $\varepsilon_{a,P}$, depends on $\varepsilon_a$ and the accuracy of evaluating the equation of $P$. Certainly, $a$ is distant from $P$ if the magnitude of $P(a)$ exceeds $\varepsilon_{a,P}$. Otherwise, $a$ is near $P$ and $a$ may lie on $P$. In the following discussion we drop all subscripts and collectively refer to the uncertainty estimates as $\varepsilon$.

### 3.2. Vertex Incidence Testing

One test performed when computing the intersection of two solids is testing whether vertex $v$ coincides with vertex $w$. This test can be implemented by testing whether $v$ is on each of the planes intersecting in $w$. This is not, however, equivalent to testing whether $w$ lies on the planes intersecting in $v$. In fact, the test is neither symmetric nor transitive. For this reason, the incidence rules developed below must be limited to carefully chosen situations. Indiscriminate application could yield to inconsistency and failure of the entire intersection computation.

All incidence tests first make the necessary floating point calculations. If the features in question are distant, no further action is required. If they are near then we examine adjacent features so as to obtain information on which to base a decision.

**Vertex on Plane:** We test whether vertex $v$ lies on plane $P$ as follows.

(1) If the magnitude of $P(v)$ is greater than $\varepsilon$ then $v$ is distant from $P$, and the sign of $P(v)$ determines whether $v$ is above or below $P$.

(2) Otherwise, we examine the intersection of $P$ with each edge incident to $v$. Consider an edge $e$ incident to $v$ and a vertex $w$, where $w$ is shown to be off $P$ by a recursive invocation of the vertex-on-plane test. If $e$ intersects $P$ at point $a$ far from $v$, then $v$ is not on $P$; $v$ and $w$ are on the same side of $P$ if $a$ is not in the interior of $e$; otherwise, $v$ and $w$ are on opposite sides of $P$.

(3) Finally, if step (2) fails to classify $v$ then $v$ is on $P$.

**Vertex on Edge:** First, we determine if vertex $v$ is on the defining planes, $P$ and $Q$, of edge $e$. If $v$ is on $P$ but not on $Q$, we know whether $v$ is to the left or right of $P \cap Q$ from testing whether $v$ is on $Q$. If $v$ is on both $P$ and $Q$ and is near a vertex of $e$ then we determine if $v$ lies on or between the planes defining the endpoints of $e$.

**Vertex on Vertex:** Vertex $v$ is coincident with vertex $w$ if $v$ lies on each of the three planes that define $w$.

Note that asking if $v$ is coincident with $w$ might result in a different answer than asking if $w$ is coincident with $v$. To avoid such an inconsistency we always perform this test asymmetrically, by asking if a vertex of the first solid is coincident with a vertex of the other solid. Another approach would have been to make the test symmetric by also requiring $w$ to be on each plane defining $v$. However, if the first test succeeded and the latter test failed, we would be in the situation where $v$ is on the defining planes of a vertex, but not coincident with their intersection.

Our tests are not sufficient to guarantee transitivity of vertex coincidence. However, this cannot introduce an inconsistency since the minimum separation between features on objects guarantees that at most one vertex from each object can be in the same neighborhood.

### 3.3. Edge Intersection and Face-Incidence Tests

Edge intersection and face-incidence tests arise in our intersection algorithm as follows A plane of face $f$ of solid $A$ can intersect solid $B$ in a collection of faces, edges, vertices, points in the interior of edges, and line segments in the interior of faces (see Figure 3.1), which form a cross-section of solid $B$. We must identify intersections of $f$ with this cross section. If $f$ is adjacent to another face $g$, then the cross sections induced by the planes of $f$ and $g$ must agree. For example, if we determine that an edge common to $f$ and $g$ intersects an edge of $B$ in the plane of $f$, then this common edge must intersect that same edge of $B$ in the plane of $g$. Structured in this way, three questions arise when deciding edge intersection. Given face $f$ contained in plane $P$ of solid $A$, and edge $e$ of $f$ defined by the intersection of $P$ with plane $Q$, we ask if $e$ intersects:

(1) edge $e'$ of $B$ where $e'$ is in $P$;

(2) edge $e'$ of $B$ where $e'$ is not in $P$; or

(3) cross-face edge $\bar{e}$.

A purely numerical computation can distinguish four possibilites: the edges appear to intersect in their interiors; the edges appear to intersect near a vertex of $e$, in the interior of $e'$ or $\bar{e}$; the edges appear to intersect in the interior of $e$, near a vertex of $\bar{e}$ or $e'$; or the edges appear to intersect near vertices.

A cross-face edge is by definition in $P$. To test whether edge $e'$ of $B$ lies in $P$, we merely test whether both vertices of $e'$ are on $P$.

To test whether $e$ intersects edge $e'$, where $e'$ is not in $P$, we test the intersection of $e'$ with $Q$. (Recall that $e$ is defined by $P \cap Q$.) If the computed intersection point appears to be in the interior of both $e$ and $e'$, then there is an intersection. If the computed intersection point is near a vertex $v$ of $e$, then we examine plane $R$ used to define $v$. If either one or both vertices of $e'$ lie on $R$, we can infer whether $e'$ intersects $v$ or the interior of $e$; otherwise, if the vertices of $e'$ are on opposite sides of $R$, then $e'$ intersects $v$.

To test whether $e$ intersects edge $e'$, where $e'$ is in $P$, we test the intersection of $e'$ with $Q$. The only new case, when $e$ and $e'$ are collinear, occurs when both vertices of $e'$ are in $Q$. In this case, $e$ and $e'$ are collinear, and it is possible that one or both of the vertices of $e$ are in the interior of $e'$. This can be resolved by a case analysis [Karasick, 1988].

Finally, we must test whether cross-face edge $\bar{e}$ intersects $e$. First we test the endpoints of $\bar{e}$ to see if they are in $Q$. (Note that these endpoints are either vertices of $B$ or points in the interior of edges of $B$.) The only new case that arises is when $\bar{e}$ is near vertex $v$ of $e$. In this case, we test the intersections of the endpoints of $\bar{e}$ with the plane $R$ used to define $v$ (by testing the vertices or edges of $B$ with $R$). If one or both of these endpoints is on $R$ then we have an intersection; otherwise, we test the vertices of $e$ against the plane of the face of $B$ that induced $\bar{e}$.

## 4. Representation

The difficulty of implementing a polyhedral solid modeler depends to a great extent on the underlying representation. There are three major choices for representing solids by their boundaries:

(1) The object's surface is restricted to be a manifold, that is, the neighborhood of each point on the surface is homeomorphic to a disc. Since coincidences of edges or vertices are prohibited, the set of representable objects is not closed under regularized set operations. In consequence, such modelers cannot handle all situations and either fail outright, or else announce an error.

(2) The object's surface is topologically a manifold, but its geometric embedding allows distinct surface points to coincide in space. This class is closed under set operations, see e.g., [Weiler, 1984].

(3) Finally, the surface is an orientable non-manifold, that is, there are points whose neighborhoods are not homeomorphic to a disc, but the surface is bounded, enclosing a possibly infinite volume. This class of objects is also closed under set operations.

Representation (1) is deemed undesirable since it is not closed under regularized set operations. As we have found, representation (2) leads to very complicated intersection algorithms and requires intricate techniques to maintain consistent topological separation when the two objects are in special position with respect to each other (see also [Paoluzzi, Ramella, and Santarelli, 1986]). Therefore, we have chosen (3) as our basic representation, for it delivers the simplest algorithms for intersecting objects in special positions.

All intersection algorithms simplify significantly when faces are restricted to triangles or convex polygons [Yamaguchi and Tokieda, 1985; Laidlaw and Hughes, 1986; Paoluzzi, Ramella, and Santarelli, 1986]. Such simplification is had at the price of increasing the number of faces. Under the best of circumstances, a space increase of 50 percent is experienced [Paoluzzi, Ramella, and Santarelli, 1986]. Another difficulty is that the intersection algorithms developed under these assumptions do not easily generalize to the curved surface domain.

We are especially concerned with limiting the redundancy of numerical data in our object representation, thereby minimizing the possibility of introducing inconsistencies when testing incidence. For this reason the only numerical data used in our representation are the coefficients of the face plane equations. Vertex coordinates are specified implicitly as the intersection of three face planes. Auxiliary planes can be used to define edges and vertices when planes of the solid boundary are nearly tangent. All other model data is given in symbolic form, e.g., which face planes intersect to define an edge, which edges are incident to a common vertex, etc.

## 4.1. Polyhedra and Solids

A *vertex* is a point in Euclidian 3-space, defined as the intersection of three planes, although we allow more than three planes to meet at a vertex. An *edge* is the line segment connecting two distinct vertices and is defined by the intersection of two planes, although we allow more than two faces to meet on an edge.

A *polyhedron* is as defined in [Preparata and Shamos, 1985], except that infinite volumes are allowed provided the surface is bounded. Many authors exclude solids with infinite volume from consideration on the grounds that such objects are not physically realizable. We do not exclude them as they are useful for intermediate steps in object design. For example, a cube with an internal void of cubic shape is conveniently understood as the intersection of two simple polyhedra, one of finite, the other of infinite volume.

The *regularized* intersection (complement, union, difference) of two polyhedra is the closure of the interior of their set-theoretic intersection (complement, union, difference) [Kuratowski and Mostowski, 1968; Requicha, 1977]. These operations constitute the regularized set operations on polyhedra. Since we consider only regularized operations, we drop the adjective.

A *solid* is either a polyhedron or it is the result of a sequence of set operations on polyhedra. In difference to polyhedra, an edge of a solid can be adjacent to many more than two faces, and each vertex can be incident to more than one corner (cycle of edge-adjacent faces). Such edges and vertices consist of surface points whose neighborhoods are not homeomorphic to a disc. When we wish to stress the presence of such non-manifold edges and vertices, we speak of *non-manifold solids*. Finally, all solids have a bounded surface.

As with polyhedra, the faces of solids must not intersect except in vertices and edges. This necessitates a general notion of a *face* that is almost the two-dimensional analogue of a solid: a face of a solid is a planar, bounded, connected, regular set.[3] The *boundary* of a face is the collection of edges and vertices that describe the intersection of the face with all other faces in the representation. This boundary is organized into a collection of *isolated vertices* and clockwise *bounding cycles* of directed edges. These directed edges (called facet/edge pairs by Dobkin and Laszlo [1987]) are orientations of edges on incident faces. Each directed edge is oriented so that in traversing a cycle of a face, the interior of the face is locally to the

[3] A set is regular iff it is equal to the closure of its interior. Note that faces are

right. (Examples of legal and illegal faces are shown in Figure 4.1.)

Associated with each directed edge is a *tangent* vector that corresponds to the direction in which the edge is traversed, and a *face-direction vector* that points orthogonally from the interior of the directed edge into the interior of the directed edge's face (this vector is the cross product of the directed-edge tangent and the face normal; see Figure 4.2). The directed edges incident to a vertex on a face can be sorted radially around that vertex by sorting their tangent vectors on their face, and the directed edges associated with an edge can be sorted radially around their edge by sorting their face direction vectors in a plane perpendicular to their edge.

The *shell* of a solid is a connected component of the surface of the solid. The representation for a solid is presented as a list of representations, one for each shell. Each shell representation is presented by lists of the faces, edges, and vertices of that shell. Our representation, called the *Star-Edge* representation, is described by Karasick [1988]. (Related representations are described by Hanrahan [1985] and Dobkin and Laszlo [1987]).

### 4.2. Reduction to Intersection and Complement

It is well known that all Boolean set operations can be reduced to intersection and complement. For the Star-Edge representation, a solid is complemented by complementing each of its shells, and a shell is complemented by inverting the normal vector to each face, the orientation of each directed edge, and the radial order of the directed-edge lists for each vertex on a face. Thus regularized set operations, such as union and difference, can be efficiently implemented in terms of intersection.

### 4.3. Ordering Operations and Pairing

Our regularized intersection algorithm makes use of some basic operations that order points along lines, find the radial order of vectors about a point in a plane, and pair certain edges or faces based on their orientation. These operations require careful implementation for the sake of robustness, but differ from incidence tests in that numerical uncertainty cannot arise in a fundamental way. For example, two consecutive points to be ordered along a line might arise as the intersection of two edges in a face with this line. If these points are near each other, then the minimum feature separation of

---

planar, connected, regular sets, and solids are regular sets in three-space.

the input implies the existence of a common vertex, and these points can be ordered by examining the radially ordered directed edges at this common vertex.

When directed edges belonging to the same face are radially ordered about a common vertex, they can be paired such that two consecutive directed edges, in radial order, enclose face interior. In this case we speak of an *area-enclosing pair* of edges. Similarly, the faces incident to a common edge are radially ordered about this edge, and these faces are paired so that consecutive faces enclose a wedge of solid interior. Here we speak of a *volume-enclosing pair* of faces.

## 5. Global Outline of the Intersection Algorithm

Conceptually, the algorithm to intersect solid $A$ with solid $B$ merges intersecting shells and retains or discards nonintersecting shells based on a containment test:

(1) Intersect every shell of $A$ with every shell of $B$.

(2) Merge intersecting shells into a set of shells that constitute a portion of the boundary of $A \cap B$.

(3) Add all shells of $A$ contained entirely within $B$ and add all shells of $B$ contained entirely within $A$.

### 5.1. Merging Shells

By far the most complex step in this algorithm is intersecting and merging shells. Conceptually, shells are intersected by intersecting their faces. If no intersections are found, the shells do not intersect. If there are intersections, then the shells are merged as required.

Assume that a shell of $A$ and a shell of $B$ intersect and must be merged. In the standard approach, one proceeds as follows:

(1) Intersect all faces of $A$ with all faces of $B$, determining the correct subdivision of the appropriate faces.

(2) Add to the subareas on the surface of $A \cap B$ the faces of $A$ that are in the interior of $B$ and the faces of $B$ that are in the interior of $A$.

Recall that the numerical computations needed to implement these steps are asymmetric: A vertex of $A$ may lie on every plane defining a vertex of $B$ but not vice versa. Therefore, it is advantageous to restructure the merging step slightly by introducing a corresponding asymmetry as follows:

(1) For each face $f$ of $A$, intersect its plane $P$ with $B$ yielding a cross-face graph, called $G_P$. Classify the areas into which $G_P$ partitions $P$ as being inside, on the surface, or outside of $B$.

(2) Intersect $G_P$ with the boundary of $f$, and determine which areas on $P$ are faces of $A \cap B$.

(3) Add all faces of $A$ that are in the interior of $B$.

(4) Transfer the relevant edges of subareas of $f$ bounding $A \cap B$ to the corresponding faces of $B$, thereby subdividing the faces of $B$ that intersect the surface of $A$.

(5) Add all faces of $B$ that are in the interior of $A$.

This approach can be interfaced with heuristics that quickly reject nonintersecting face-pairs (e.g., [Laidlaw and Hughes, 1986]), and combined with techniques from computational geometry to yield an asymptotically efficient algorithm [Mehlhorn, 1984]. Briefly, each face is boxed. The set of boxes is intersected to determine a subset of face-pairs that might intersect. The time required to find all intersecting boxes is $O(n \log^2 n + k)$ where $n$ is the number of boxes and $k$ is the number of box-pair intersections.

## 5.2. Handling Nonintersecting Shells

It is possible that some shells of $A$ and $B$ do not intersect. For each such shell of $A$ we must determine whether it is in the interior of $B$, and similarly for each such shell of $B$ we must determine whether it is in the interior of $A$. The four possible outcomes are:

$$A \cap B = \begin{cases} A & \text{shell}(B) \subset A \ \text{but shell}(A) \not\subset B \\ B & \text{shell}(A) \subset B \ \text{but shell}(B) \not\subset A \\ U(A,B) & \text{shell}(B) \subset A \ \text{and shell}(A) \subset B \\ \varnothing & \text{otherwise} \end{cases}$$

where $U(A,B)$ is the solid bounded between the shells of $A$ and $B$. To implement this containment test, select a point on the surface of $A$ and a point on the surface of $B$ and classify the connecting line as described in [Requicha and Voelcker, 1985]. A numerically robust method for implementing this computation is described in [Karasick, 1988].

### 5.3. Surface Adjustments

Recall that all objects to be intersected have no features smaller than a given tolerance, say $\varepsilon$. After a set operation, however, the resulting object may well have such features, and in a post-processing step one might wish to adjust its surfaces so as to eliminate them. Such an adjustment must include, among other things, the elimination of short edges, called $\varepsilon$-edges.

Suppose that $\varepsilon$-edge $e$ exists. If one of the vertices adjacent to $e$ is of degree three then $e$ can be removed by tilting the third face incident to that vertex. If the face has at most two vertices of degree four or higher, then the face can be rotated about the line through these two high-degree vertices without creating new $\varepsilon$-edges. Otherwise, if the face has at most three high-degree vertices one of which is of degree four, then tilting the face about the other two high-degree vertices transfers the $\varepsilon$-edge to a new set of faces, from which it might be removed. We are presently studying ways to eliminate $\varepsilon$-edges.

## 6. The Cross-Sectional Graph ($G_P$)

We describe the construction of the graph $G_P$ that comprises the intersection of solid $B$ with the plane ($P$) containing face $f$ of solid $A$. Briefly, edges of $B$ are intersected with $P$, yielding *intersection points*. Certain intersection points are then linked by line segments representing the intersection of faces of $B$ with $P$. The resulting graph ($G_P$) partitions $P$. The areas delimited by its edges and vertices consist of points that lie inside, outside, or on the surface of $B$. Such areas could then be labeled as **in**$B$, **out**$B$, or **on**$B$. If an area delimited by edges and vertices of $G_P$ is on the surface of $B$, then we distinguish whether on $B$ this surface area is oriented in the same way as $P$. If the orientation is opposite, then the area cannot be part of the surface of $A \cap B$. Accordingly, the label **on**$B$ could be refined to $\mathbf{on}_{\mathbf{in}}B$ for areas oriented the same way, and $\mathbf{on}_{\mathbf{out}}B$ for areas oriented the opposite way. This area classification is effected by suitably orienting the directed edges of $G_P$, ignoring the distinction between **in**$B$ and $\mathbf{on}_{\mathbf{in}}B$.

The construction of $G_p$ requires many incidence tests. It is of critical importance to robustness that the outcome of a test be the same if this test is repeated when constructing a graph for an adjacent face. This is achieved by annotating the data structures for $G_p$ and constructing these graphs together.

### 6.1. Graph Construction

Each edge of $B$ is intersected with plane $P$ of $A$. Then the line segments that are intersections of $P$ with faces of $B$ are determined. Face $g$ of $B$ can either lie on $P$ or intersect $P$ in a set of segments of the line $l$ that is the intersection of $P$ with the plane of $g$. If an edge of $B$ lies on $P$, its vertices are considered to be the intersection points. If $g$ does not lie on $P$, then we sort the intersection points of $g$ with $P$ along $l$ and consider them in order. For each intersection point we determine whether there is a line segment (called a *cross-face edge*) connecting that point with the next point in order. Such a line segment represents a component of the intersection of $g$ with $P$. Note that two intersection points along $l$ may be close together if they are incident to a vertex of $B$ that is near but not on $P$. Here the relative order is resolved by examining the radial order of these edges at their common vertex.

A cross-face edge is created beginning at an intersection point if line $l$ points into the interior of face $g$. Let $t$ be a tangent vector corresponding to the order of the intersection points along $l$, and let $a$ be such an intersection point. If $a$ is in the interior of an edge of $g$ and this edge has a directed edge on $g$ whose face-direction vector has a positive dot-product with $t$, then we begin a cross-face edge (see Figure 6.1) If $a$ is at a vertex of $g$, insert $t$ into the radially ordered list of directed edges at that vertex. We begin a cross-face edge if $t$ splits an area enclosing pair of directed edges (see Figure 6.2).

### 6.2. Directed-edge Orientation

The edges of $G_P$ are either edges of solid $B$ or they are cross-face edges across faces of $B$. We now consider how to orient the directed edges of an edge of $G_P$ in order that we may construct the bounding directed-edge-cycles for $G_P$. A cross-face edge of $G_P$ gets a single directed edge, oriented so that the interior of $B$ is to the right (see Figure 6.3[a]). An edge of $G_P$ induced by edge $e$ of $B$ gets one or two directed edges using the following idea. If on one side of $e$, $P$ is locally in the interior of $B$ (**in**$B$ or $\mathbf{on}_{\mathbf{in}}B$), then create a directed edge (compute face-direction vectors for the left and right sides of $e$ on $P$, and determine if each vector splits a volume-enclosing pair of faces adjacent to $e$). If we find that $P$ is locally in the exterior of $B$ on both sides of $e$ (**out**$B$ or $\mathbf{on}_{\mathbf{out}}B$), then $G_P$ gets only the vertices of $e$ (see Figure 6.3[b]); if $P$ is locally in the interior of $B$ on both sides of $e$, then we create two oppositely-oriented directed edges (see Figure 6.3[c]); finally, if $P$ is locally in the interior of $B$ on one side of $e$, then we create a directed edge on $G_P$, oriented with the **in**$B$ cycle to the right (see Figure 6.3[d]).

### 6.3. Classification of Isolated Vertices

By orienting the directed edges of $G_P$, the procedure of the previous section implicitly classified all regions on $P$. Isolated vertices can be components of the boundary of either an **in**$B$ or an **out**$B$ region. Classification of isolated vertices is more complicated than classification of directed edges, and is described in [Karasick, 1988].

## 7. Subdividing Faces of Solid $A$

We must subdivide a face of solid $A$ in order to identify those subareas that are on the surface of $A \cap B$. The subdivision is effected by intersecting the boundary of that face with the associated cross-sectional graph. (Recall that $G_P$ is the cross-section of solid $B$ with plane $P$ of face $f$.) The areas bounded by $G_P$ are a set of faces or the two-dimensional regularized complement of such a set. Intersecting $G_P$ with $f$ therefore has the flavor of an intersection algorithm of polygonal, two-dimensional objects, and many of the steps to be described here are analogous to the entire three-dimensional intersection algorithm. The structural analogy with polyhedral intersection is seen by equating edge cycles with shells.

Conceptually, we intersect the areas of $G_P$ labeled **in**$B$ or $\mathbf{on_{in}}B$ with $f$. Since we cannot work directly with polygonal areas, we must express this computation in terms of operations on the bounding directed-edge-cycles of $f$ and $G_P$. For this reason we oriented the edges of $G_P$. Roughly speaking, the intersection of $f$ with $G_P$ is done as follows:

(1) Intersect the edges of $f$ with the edges of $G_P$.

(2) At each intersection point, determine which incident edge segments bound faces of $A \cap B$.

(3) Construct bounding directed-edge-cycles of $A \cap B$ by traversing directed edges of $G_P$ and $f$ between intersection points.

(4) Add all additional edges (directed edges and vertices) arising from contained, nonintersecting components of $G_P$ and $f$.

### 7.1. Intersection Analysis

The edges and vertices of $G_P$ are intersected with the edges and vertices of $f$ resulting in a set of intersection points. If two edges overlap, then the intersection of the vertices of one edge with the other edge (and vice versa) constitute the intersection points. All edges are subdivided by their intersection points. For example, two edges intersecting in interior points become four edges, and directed-edge segments inherit orientations.

At every intersection point the incident directed-edges are ordered radially. Then these directed edges are examined twice. In the first pass, all directed edges of $f$ that lie within an area of $G_P$ inside $B$ are marked; in the second pass, all directed edges of $G_P$ that lie within $f$ are marked. All unmarked directed-edges are then discarded. Coincident directed edges are treated as directed edges of both $f$ and $G_P$. Finally, remaining radially-adjacent directed-edges which enclose area are paired and bound the same face of $A \cap B$.

For example, at the intersection point shown in Figure 7.1 directed edges 1 through 6 belong to $G_P$, directed edges 7 through 12 to $f$. Directed edges 7 and 10 of $f$ are within a cycle of $G_P$ enclosing area inside $B$. Directed edges 2, 3, 5 and 6 are inside $f$. All other directed edges are discarded. The remaining directed edges are paired according to whether they enclose area.

### 7.2. Cycle Structure

Some of the cycles bounding a face of $A \cap B$ are the result of intersecting the edge of face $f$ of solid $A$ with the edges of $G_P$. The remaining bounding cycles, if any, are either cycles of $f$ contained in some area enclosed by cycles of $G_P$, or they are cycles of $G_P$ contained in $f$. For example, consider the cycles of Figure 7.2. Here the face $f$ is bounded by $C_1$ and $C_2$, and $G_P$ consists of two areas, one enclosed by $C_6$, the other by $C_3$, $C_4$, and $C_5$. All cycles are oriented as shown. $C_1$ and $C_3$ intersect and yield the merged cycle (1,2,3,4). $C_4$ and $C_5$ both are within $f$ and so must bound some area of intersection. $C_6$ is not contained within $f$ and will be discarded. $C_2$, a nonintersecting cycle of $f$, is not within any enclosed area of $G_P$ and is also discarded. As result, the intersection area is bounded by the cycles (1,2,3,4), $C_4$, and $C_5$.

When testing whether a cycle $c$ lies within the area enclosed by group **C** of cycles we know that these cycles do not intersect each other. Conceptually, we pick a point $u$ on $c$ and a point $v$ on one of the cycles of **C**. We then intersect the ray directed from $u$ through $v$ with the cycles of **C**. This partitions that ray into segments that are inside or outside the area bounded by the cycles of **C**. For example, in Figure 7.3 we have the intersection points 1, $v$, and 2. The segments $(u, 1)$ and $(v, 2)$ are inside the enclosed area, the other two segments are not. Finally, we ask whether a segment containing $u$ is an inside segment.

### 7.3. Closure on Faces of $A$

It is possible that a face of solid $A$ is in $A \cap B$, even though the corresponding graph $G_P$ is empty. Assume that $f$ is such a face, and further that the shell of $A$ to which $f$ belongs intersects the surface of $B$. Face $f$ can be added to $A \cap B$ because it is reachable by vertex and edge adjacent faces from another face $f'$ of $A$ with a nonempty corresponding graph $G_{P'}$.

## 8. Adding Faces Induced by $B$

When the processing of the previous section is complete, all faces of $A \cap B$ that lie on the surface of $A$ have been generated. The final step in the algorithm is the subdivision of faces of $B$, i.e., we add to $A \cap B$ all missing faces that lie on the surface of $B$. The simplest way to obtain those faces is to run the algorithm with the role of $A$ and $B$ interchanged, but a more direct approach results in greater robustness.

First we examine the edges and vertices of $A \cap B$. We identify those that are adjacent to each face $g$ of $B$ extending inside $A$. That is, every intersection point or line between $g$ and a face of $A$ is transferred to $g$.

As an intersection line is transferred to $g$, its neighborhoods are classified (on $g$) as **in**$A$ or **out**$A$. As in Section 6, the neighborhood classification is expressed through directed-edge orientation. Also transferred and classified are isolated vertex intersections (isolated vertices of some $G_P$). Finally, all faces of $B$ must be examined and the relevant face areas on those faces are assembled into faces of $A \cap B$. Note that areas that lie on the surface of both $A$ and $B$ can be ignored, as they were classified as $\mathbf{on}_{\mathrm{in}}B$ and have already been accounted for.

### 8.1. Intersection Line Transfer

We consider all directed edges of $A \cap B$ that are contained in face $f$ of $A$. As before, $P$ is the plane containing $f$. An edge in $P$ and in a face of $A \cap B$ is:

(1) an edge segment of $f$ located in an **in**$B$ area of $G_P$, or

(2) an edge segment of $f$ located in an $\mathbf{in_{on}}B$ area of $G_P$, or

(3) a cross-face edge segment of $G_P$ not coincident with an edge of $f$, or

(4) an edge segment of $B$ not coincident with any edge of $f$, or

(5) an edge segment of $G_P$ coincident with an edge of $f$.

All edges of type (1) may be ignored, for no face of $B$ intersects in that edge. Edge segment $e$ of type (2) is also located on a face $g$ of $B$, and $e$ is transferred to $g$ by creating a directed edge of $e$ on $g$ with orientation opposite to the directed edge of $e$ already created. Cross-face edge-segment $\overline{e}$ of type (3) is a portion of the intersection of $g$ with $P$, and it is added to $g$ by creating a directed edge of $\overline{e}$ on $g$ with opposite orientation to that directed edge of $\overline{e}$ already created. Edge segment $e$ of type (4) is transferred to each face of $B$ adjacent to $e$ and below $P$ (see Figure 8.1). Finally, an edge segment of type (5) is transferred to all those faces of $B$ that lie between volume-enclosing face pairs of the coincident edge of $A$. (In Figure 8.2, face $g$ is inside $A$, whereas face $h$ is not.) This transfer is done by merging the radially-ordered directed-edges of the two coincident edges of $A$ and $B$.

### 8.2. Isolated Vertex Transfer

The need for transferring isolated vertices is illustrated in Figure 8.3. Here the complement of tetrahedron $B$ is intersected with cube $A$. The surface of $B$ is contained entirely within $A$ except for isolated vertex $v$ that requires the faces of $B$ to be added to the surface of $A \cap B$. This is accomplished by transferring every isolated vertex contained in a known face of $A$ to $B$. There are three situations: the isolated vertex may be in a face interior, on the edge of a face, or coincident with a vertex. We sketch how the cases are handled below.

For a vertex $v$ of $B$ in the interior of a face $f$ of $A$, we determine which edges of $B$ incident to $v$ are inside $A$. Vectors directed from $v$ along these edges have negative dot products with the normal of $P$. Vertex $v$ is transferred to all faces of $B$ adjacent to such edges. For a vertex $v$ of $B$ in the interior of an edge segmentof $A$, we determine if an edge $e$ of $B$ incident to $v$ is inside $A$. If $e$ lies between volume-enclosing pairs of faces adjacent to the edge segment of $A$ (see Figure 8.4.), then $v$ is transferred to faces adjacent to $e$. For a vertex $v$ of $B$ coincident with vertex $w$ of $A$, we can determine if an edge $e$ of $B$ incident to $v$ is in the interior of $A$ as follows. We construct a plane $Q$ containing both $e$ and an edge of $A$ incident to $v$; we construct directed lines on $Q$ which represent local intersections of $Q$ with faces of $A$ incident to $v$. Finally, $e$ is in the interior of $A$ if $e$ splits an area-enclosing pair of directed lines on $Q$ (see Figure 8.5), and $v$ is transferred to the faces of $B$ adjacent to $e$.

### 8.3. Subdivision and Closure

After all relevant edges and vertices have been transferred, the respective faces of $B$ are subdivided with the techniques of Section 6. The resulting faces of $A \cap B$ are bounded by cycles consisting of transferred edges and vertices, as well as edges and vertices of $B$ entirely inside $A$. All faces of $B$ that are adjacent to these edges or vertices and are not subdivided must therefore also lie in the interior of $A$ and must be added as faces of $A \cap B$. These faces are discovered by breadth-first search.

At this stage in the algorithm, all faces of $A \cap B$ that are the result of merging shells have been constructed. The faces obtained may need to be subdivided so that their interior is connected. This is a routine step involving an analysis of bounding-cycle containment. After legal faces have been obtained, a graph traversal is done to determine whether the resulting surface consists of several shells.

## 9. References

[1] Baumgart, B.G., *Winged-Edge polyhedron representation*, Rep. CS-320, Stanford A.I. Lab, Stanford Univ., Stanford, CA., 1972.

[2] Boyse, J.W., Rosen, J.M., "GMSOLID - Interactive Modeling for Design and analysis of Solids," *IEEE Computer Graphics and Applications*, March, 1982, pp. 27-40.

[3] Braid, I.C., "The Synthesis of Solids Bounded by Many Faces," *Comm. ACM*, April, 1975, pp. 209-216.

[4] Brown, C.M., "PADL-2: A Technical Summary," *IEEE Computer Graphics and Applications*, March, 1982, pp.69-84.

[5] Dobkin, D.P., Laszlo, M.J., "Primitives for the Manipulation of Three-Dimensional Subdivisions," *Proc. 3rd Symp. on Computational Geometry*, Waterloo, Ont., June 1987, pp. 86-99.

[6] Durrant-Whyte, H.F., "Concerning Uncertain Geometry in Robotics," *International Workshop on Geometric Reasoning*, Oxford, England, 1986.

[7] Eastman. C.M., Preiss, K., "A review of solid shape modeling based on integrity verification," *Computer Aided Design, March 1984, pp. 66-80.*

[8] Forsythe, G., Moler, C.B., *Computer Solution of Linear Algebraic Systems*, Prentice-Hall, 1967.

[9] Hanrahan, P.M., *Topological Shape Models*, Ph.D. Thesis, University of Wisconsin, Madison, WI, 1985.

[10] Hillyard, R., "The Build Group of Solid modelers," *IEEE Computer Graphics and Applications", March, 1982, pp. 43-52.*

[11] Hoffman, C.M., Hopcroft, J.E., "Simulation of Physical Systems from Geometric Models," *IEEE J. on Robotics and Automation*, June 1987, pp. 194-206.

[12] Karasick, M., "On the Representation and Manipulation of Rigid Solids," Ph.D. Thesis, McGill University, 1988.

[13] Kuratowski, K., Mostowski, A., *Set Theory*, North-Holland, Amsterdam, 1968.

[14] Laidlaw, D.H., Hughes, J.F., "Constructive Solid geometry for Polyhedral Objects," Brown Univ, Providence, RI, 1986.

[15] Malenkovic, V.J., "Verifiable Implementations of Geometric Algorithms Using Finite Precision Arithmetic," *International Workshop on Geometric Reasoning*, Oxford, England, 1986.

[16] Mantyla, M., "A Note on the Modeling Space of Euler operators," *Computer Vision, Graphics, and Image Processing*, Jan. 1984, pp.45-60.

[17] Mantyla, M., "Boolean Operations of 2-Manifolds through Vertex Neighbourhood Classification," *ACM Trans. Graphics*, Jan. 1986, pp. 1-29.

[18] Mehlhorn, K., *Data Structures and Algorithms*, vol. 3, Springer Verlag, New York 1984.

[19] Mortenson, M.E., *Geometric Modeling*, John Wiley, New York, NY, 1985.

[20] Owen, J., Oral communication describing the capabilities of Romulus, 1986.

[21] Paoluzzi, A., Ramella, M., Santarelli, A., "Un modellatore geometrico su rappresentazioni *triango-alate*," Rap. 13.86, Disp. Informatica e Sistemistica, Univ. Rome, Rome, Italy., 1986.

[22] Preparata, F.P., Shamos, M.I., *Computational Geometry: An Introduction*, Springer Verlag, New York, NY, 1985.

[23] Requicha, A.A.G., "Mathematical Models of rigid solids objects," Tech. Memo 28, Production Automation Project, Univ. of Rochester, Rochester, NY, 1977.

[24] Requicha, A.A.G., Voelcker, H.B., "Constructive Solid Geometry," Tech. Memo 25, Production Automation Project, Univ. of Rochester, Rochester, NY, 1977.

[25] Requicha, A.A.G., Voelcker, H.B., "Boolean Operations in Solid Modeling: Boundary Evaluation and Merging Algorithms," *Proc. IEEE*, January, 1985, pp. 30-44.

[26] Weiler, K., "Topology as a Framework for Solid Modeling," *Proc. Graphics Interface 84*, Ottawa, Canada, 1984.

[27] Wesley, M.A., Lozano-Perez, T., Lieberman, L.I., Lavin, M.A., Grossman, D.D., "A Geometric Modeling System for Automated Mechanical Assembly," *IBM J. Res. and Dev. 24*, 1980, 64-74.

[28] Yamaguchi, F., Tokieda, T., "Bridge Edge and Triangulation Approach in Solid Modeling," in *Frontiers in Computer Graphics*, Springer Verlag, New York 1985.

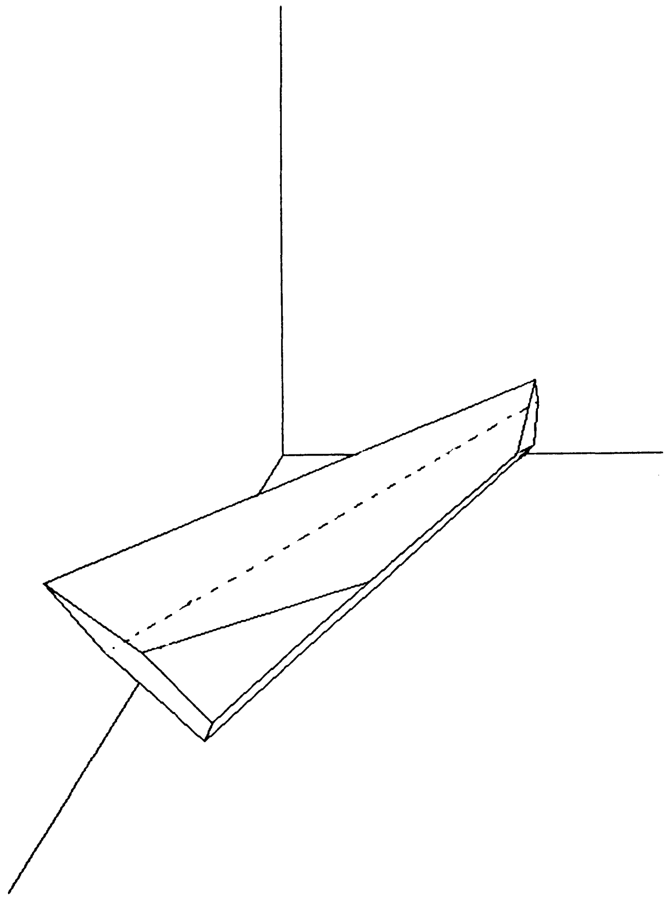

Figure 2.1

Test object was rotated about the origin to obtain a second object, which was intersected with the first.

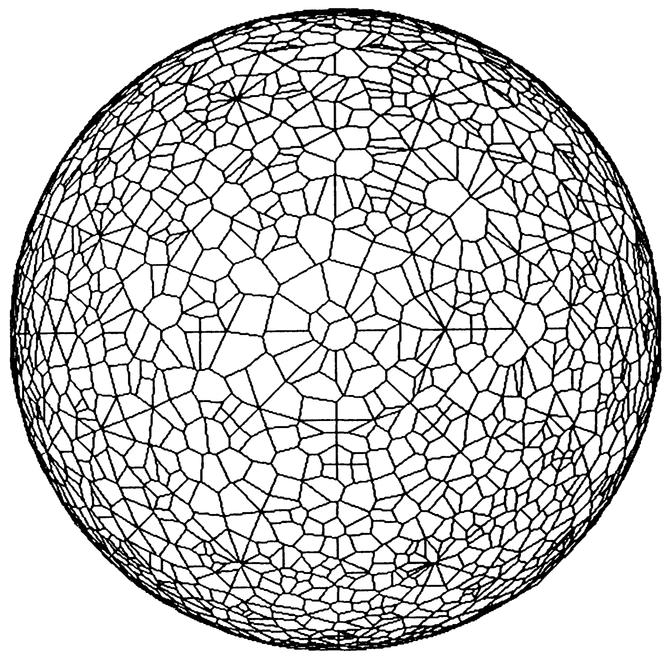

Figure 2.2

Polyhedral approximation to a sphere

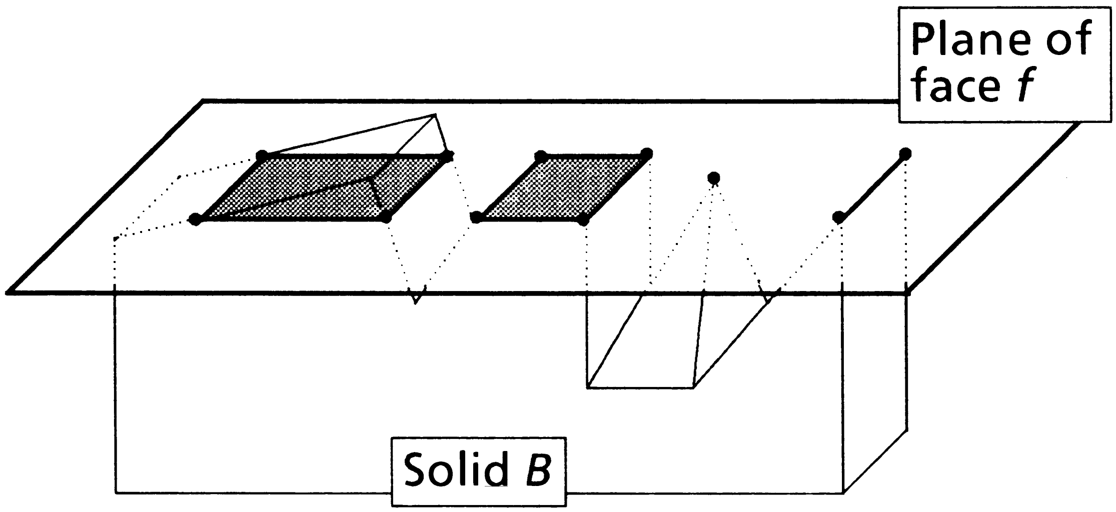


Figure 3.1

Cross section of solid $B$ with the plane of face $f$

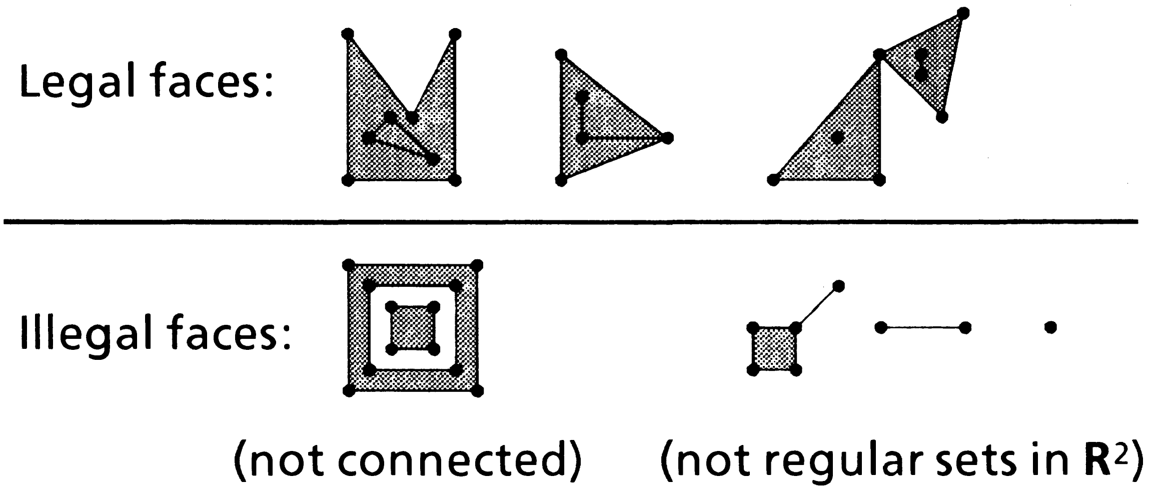


Figure 4.1

Legal and illegal faces

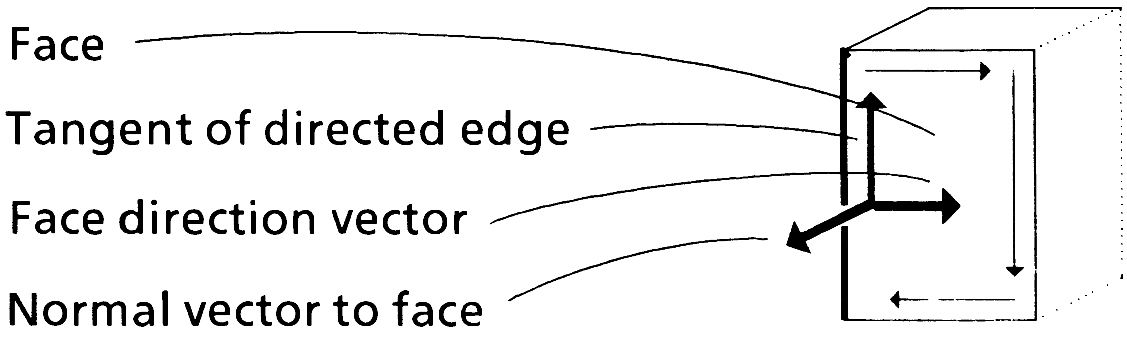


Figure 4.2

Tangent and face-direction vectors of a directed edge

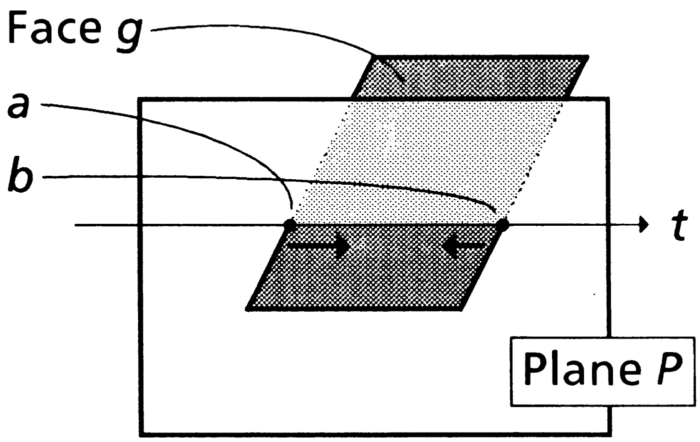


Figure 6.1

Face-direction vector at *a* has a positive dot-product with *t*; face-direction vector at *b* has not.

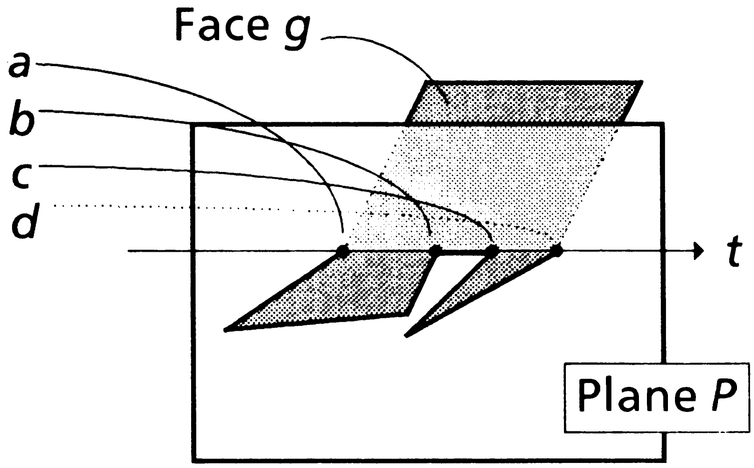


Figure 6.2

Vector *t* splits an area enclosing pair of directed edges only at vertex *a*.

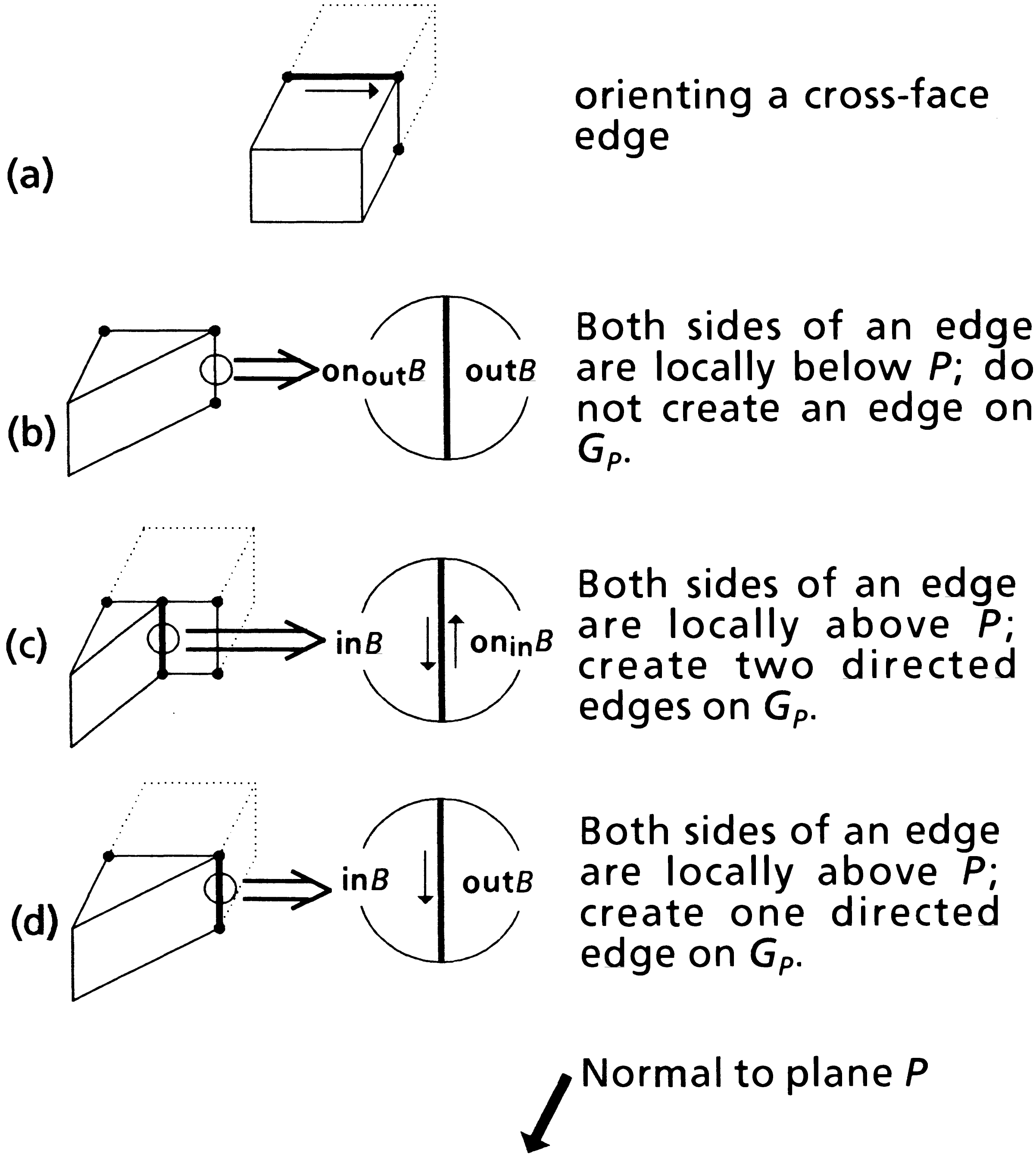


Figure 6.3

Directing edges on $G_P$

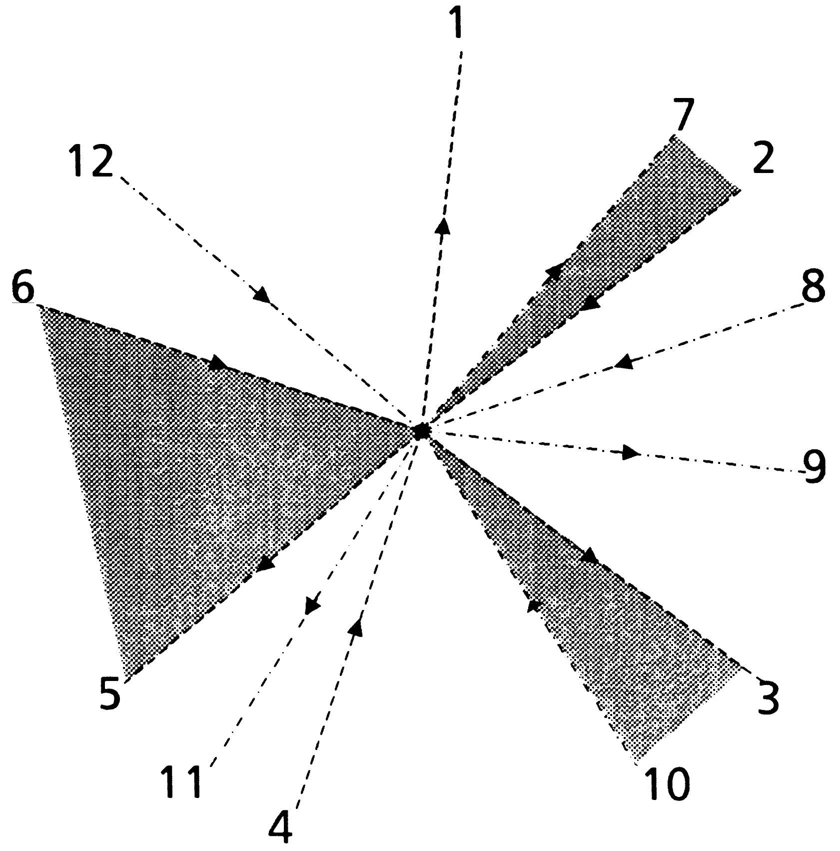


Figure 7.1

Intersection-point analysis at coincident vertices

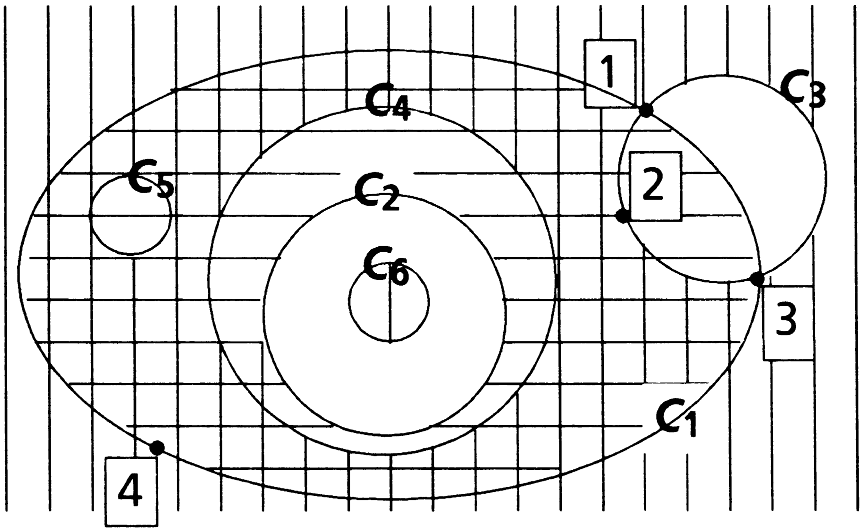


Figure 7.2

Analysis of containment: $C_3$ and $C_1$ intersect, $C_4$ and $C_5$ are contained by $C_1$.

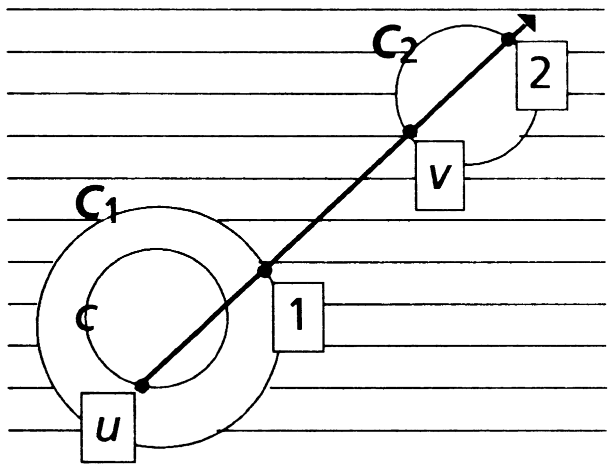


Figure 7.3

Testing for containment of $C$ in the area bounded by $C_1$ and $C_2$.

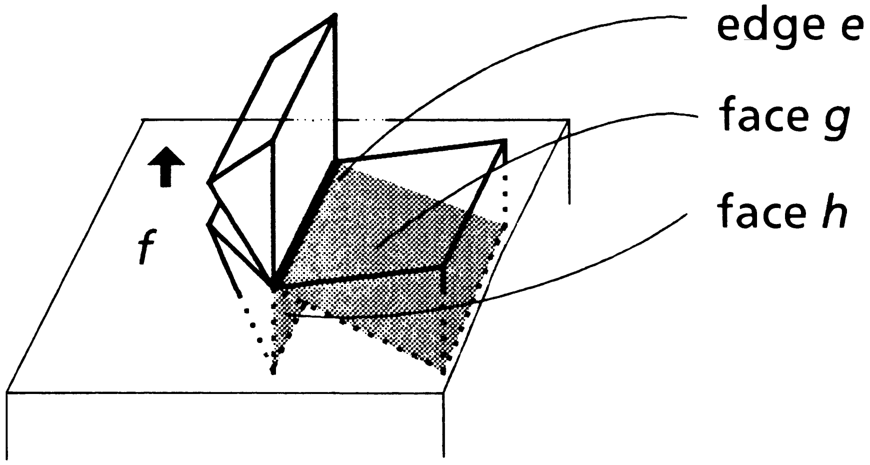


Figure 8.1

Edge e is transferred to faces *g* and *h* of solid *B* below *f*

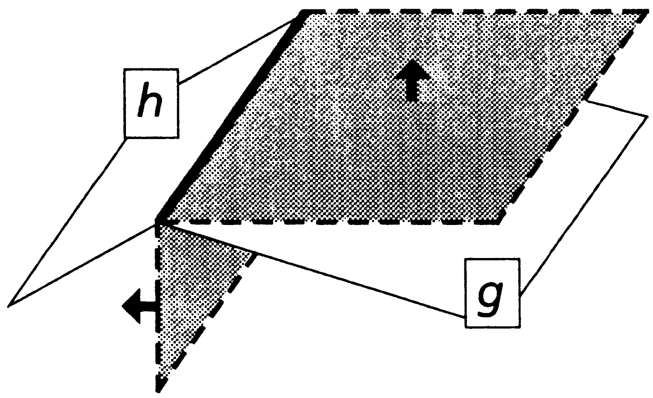


Figure 8.2

Face *g* of solid *B* is contained in a volume-enclising pair of faces of solid *A*; face *h* is not.

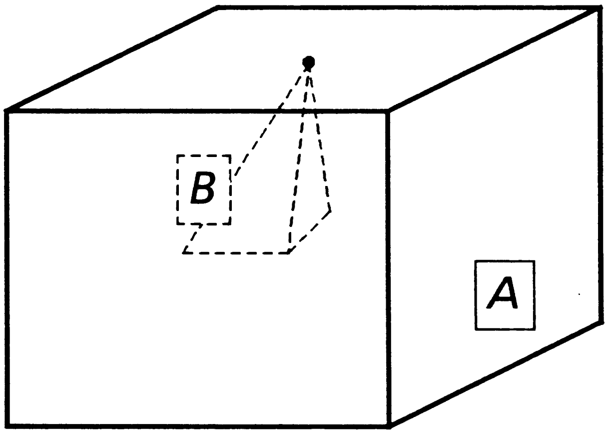


Figure 8.3

Solid *A*, a cube, is intersected with solid *B*, the complement of a tetrahedron. Their surfaces have a single point in common

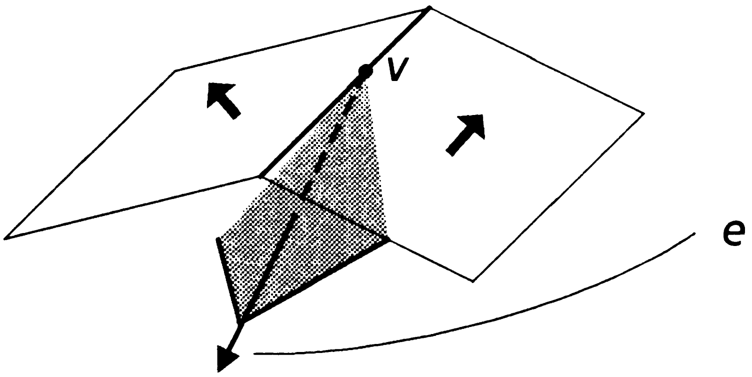


Figure 8.4

Edge *e* of solid *B* lies between a volume-enclosing pair of faces adjacent to an edge segment of solid *A*, and isolated vertex *v* is transferred to the two faces adjacent to e.

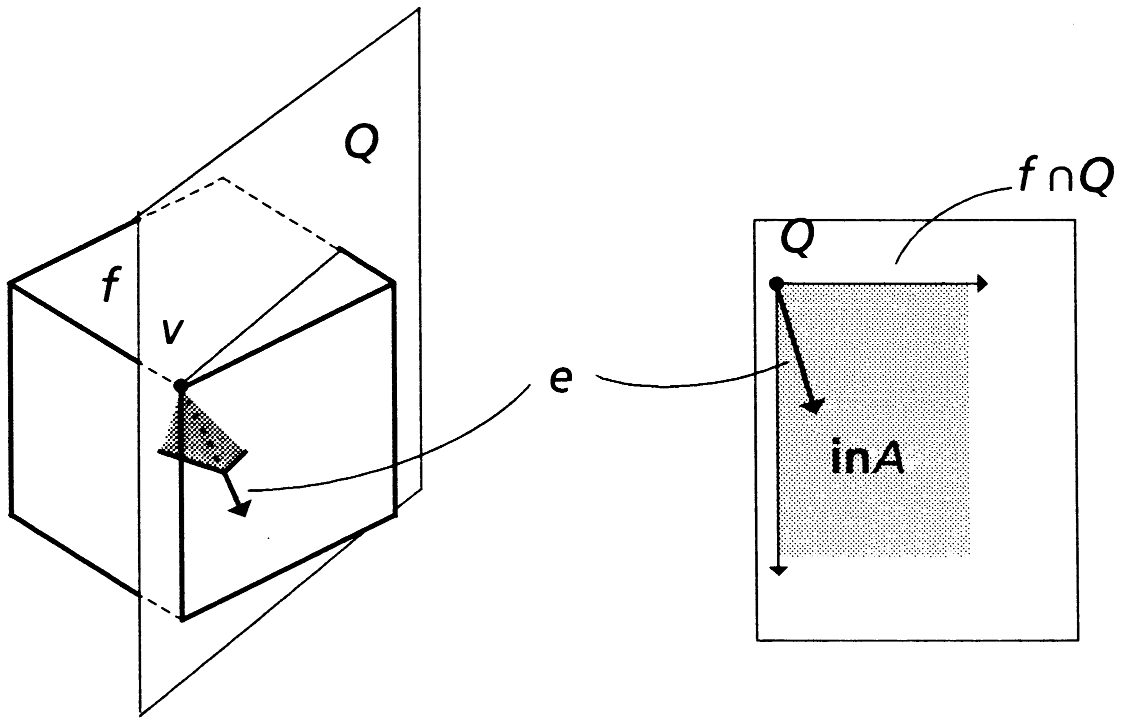


Figure 8.5

Plane Q contains edge *e* of *B* and an edge of *A* incident to *v*. Edge *e* splits an area-enclosing pair of directed lines incident to *v*, and so *v* is transferred to all faces of *B* adjacent to *e*.