# On the Representation and Manipulation of Rigid Solids

Michael Karasick
Ph. D. Thesis

TR 89-976
March 1989

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

# ON THE REPRESENTATION AND MANIPULATION OF RIGID SOLIDS

Michael Karasick

Department of Computer Science
McGill University, Montreal
*November 15th, 1988*

A thesis

Submitted to the Faculty of Graduate Studies and Research

in partial fulfillment of the Requirements for the Degree of

Doctor of Philosophy

# Abstract

Solid modeling studies how to represent geometric properties of solids by computer. A fundamental operation is the construction of representations of solids. Algorithms for set operations construct boundary representations of solids from boundary representations of other solids.

A correct and efficient intersection algorithm for polyhedral solids that uses boundary representations is described. A finite-precision implementation of the algorithm uses incidence tests that use symbolic inference in order to limit errors due to finite-precision approximations. The incidence tests are described and experimental evidence is presented to show that the incidence tests are both empirically reliable and practical.

The intersection algorithm uses a new boundary representation called the Star-Edge representation. A complementation algorithm for solids that uses the new representation is given, and an algorithm is given that uses the new representation to determine if two boundary representations describe the same solid. A canonical boundary representation for solids is described and used to prove a lower bound for the same-object problem.

# Résumé

La modélisation volumétrique est l'étude de la représentation des propriétés géométriques des objets solides. Une des opérations fondamentales est la construction d'une représentation d'objects. Les algorithmes pour les opérations d'ensembles (union, intersection et complémentation) construisent la représentation d'un objet à partir des représentations des composents de l'objet principal.

Cette thèse présente une nouvelle représentation par des contours fermés (boundary representation) des objets polyhédraux, appelée le Star-Edge, et trois algorithmes basés sur cette représentation. Le premier algorithme calcule l'intersection d'objets polyhédraux. Cet algorithme est correct et efficace. Dans l'implémentation de l'algorithme, l'utilisation d'inférences symboliques permet de réduire les erreurs introduites par les approximations de l'arithmétique de précision finie. On présente des résultats experimentaux qui indiquent que l'implémentation de l'algorithme d'intersection est à la fois pratique et fiable.

Aussi décrits sont un algorithme de complémentation des objets et un algorithme qui permet de déterminer si deux représentations décrivent le même objet. Utilisant une représentation canonique, on démontre une limite inférieure de complexité du probléme de l'égalité de deux représentations.

# Acknowledgements

A thesis is rarely written in a vacuum, and this one is no exception. Many people have contributed either implicitly or explicitly through discussion, collaboration, or moral support.

My collaborators, Chris Hoffmann and John Hopcroft, provided seemingly unbounded quantities of insight, patience, intuition, and determination. My office-mate at Cornell, Jim Cremer, gave friendship, moral support, and facility with a LISP machine that made the project much less daunting and, even occasionally, enjoyable. Discussions with Chanderjit Bajaj, Alberto Paoluzzi and Allen Back helped me understand what I was supposed to be doing. Donna Patteson and Vicki Feinberg smoothed the way.

I would also like to thank my supervisor at McGill, Sue Whitesides, for introducing me to John Hopcroft and for her careful and insightful readings of drafts of this thesis. Working on a thesis 300 miles away from a supervisor provided a few logistical difficulties. These difficulties were lessened by Charles Snow and his friendship, couch, and Glenlivet. I found my visits to Montreal to be all too brief.

Many people, both at Cornell and far away, read drafts of this thesis. Joe Mitchell read much of it at least once, as did Dave Strip, Brad Vander Zanden, Margot Reasner, and Allen Back.

I would also like to thank several granting agencies. While at McGill, I was funded by the National Sciences and Engineering Research Council. While at Cornell I was funded by the Office of Naval Research and the National Science Foundation. I would also like to thank IBM for allowing me the time necessary to finish this thesis.

On a more personal level, many friends both in Ithaca and in Montreal were very supportive. Certainly the "AI" and "Structural Complexity" seminars were welcome diversions. Bruce Donald supplied both moral support and a hidden-line algorithm used to render some of figures in this thesis. I would like to thank Stacey for gently tolerating my pigheadedness. Finally, my parents have unconditionally encouraged and supported me from beginning to end.

# Table of Contents

**Chapter 1. Introduction** . . . . . **1**
1.1 Defining Solids . . . . . 1
1.2 Why study the intersection of solids? . . . . . 3
1.3 Representing Solids . . . . . 5
1.4 Intersecting Solids Represented by their Boundaries . . . . . 7
1.5 Numerical Robustness of Geometric Algorithms . . . . . 8

**Chapter 2. Representing Solids** . . . . . **12**
2.1 A Minimal Boundary Representation for Solids . . . . . 16
2.2 The Star-Edge Boundary Representation . . . . . 19
2.3 Some Combinatorial Properties of the Star-Edge Representation . . . . . 22
2.4 A Survey of Boundary Representations . . . . . 24

**Chapter 3. Some Representational Results** . . . . . **29**
3.1 Regularized Complementation for Solids . . . . . 30
3.2 The Same-Object Problem for Solids . . . . . 32

**Chapter 4. Intersecting Solids** . . . . . **42**
4.1 Overview of the Intersection Algorithm . . . . . 45
4.2 Asymptotic Complexity of the Intersection Algorithm . . . . . 50
4.3 Computing Cross-Sections of Solid *B* . . . . . 51
4.4 Intersecting Faces of Solid *A* with Cross-Sections of Solid *B* . . . . . 57
4.5 Intersecting Faces of Solid *B* with Solid *A* . . . . . 65
4.6 Constructing the Star-Edge Representation of $A \cap^{*} B$ . . . . . 76
4.7 Intersecting Solids with Multiple Shells . . . . . 79
4.8 Summary . . . . . 84

**Chapter 5. Incidence Tests** . . . . . **85**
5.1 Incidence in a Finite-Precision World . . . . . 85
5.2 Simplifications and Assumptions . . . . . 87
5.3 Vertex-Incidence Tests . . . . . 90
5.4 Edge-Incidence Tests . . . . . 92
5.5 Face-Incidence Tests . . . . . 96
5.6 Sorting Intersection Points along a Line . . . . . 99

**Chapter 6. Conclusions** . . . . . **100**
6.1 Implementing the Intersection Algorithm . . . . . 101

6.2 The Star-Edge Representation . . . . . . . . . . 101
6.3 Asymptotic Complexity of the Intersection Algorithm . . . . . . . . . . 102
6.4 Future Work . . . . . . . . . . 102

**Appendix A. Some Elementary Topology . . . . . . . . . . 104**

**Appendix B. A Star-Edge Data Structure . . . . . . . . . . 106**

**Appendix C. Geometric Primitives . . . . . . . . . . 109**

**Appendix D. Benchmarks . . . . . . . . . . 112**
D.1 The Compound of Five Cubes . . . . . . . . . . 112
D.2 Perturbation Experiments . . . . . . . . . . 115
D.3 Approximating Quadric Surfaces . . . . . . . . . . 123

**Appendix E. Eliminating Extraneous Incidence Tests . . . . . . . . . . 126**

**References . . . . . . . . . . 128**

**Index . . . . . . . . . . 133**

# List of Figures


Figure 1. A solid has planar surfaces . . . . . 2
Figure 2. Regularization of a point set in $\mathbf{R}^2$ . . . . . 2
Figure 3. Regularized Intersection . . . . . 3
Figure 4. Manifold solids are a proper subset of solids . . . . . 3
Figure 5. A cell complex in $\mathbf{R}^3$ . . . . . 4
Figure 6. Representations vary widely in size . . . . . 5
Figure 7. Vertex, edge, and face . . . . . 6
Figure 8. A boundary representation of a tetrahedron . . . . . 7
Figure 9. Inconsistent finite-precision geometric computations . . . . . 9
Figure 10. Manifold-solid vertices have simple neighbourhoods . . . . . 13
Figure 11. Weiler's technique for representing solids . . . . . 14
Figure 12. Face and solid connectivity . . . . . 15
Figure 13. The neighbourhood of a point of a face . . . . . 16
Figure 14. Representational properties . . . . . 16
Figure 15. Defining a face of B-Rep$_{min}$ . . . . . 18
Figure 16. Orienting a face/edge intersection . . . . . 20
Figure 17. Directed edge notation . . . . . 21
Figure 18. The successor of a directed edge of a face . . . . . 22
Figure 19. Radial ordering of directed edges . . . . . 23
Figure 20. Adjacencies of the Winged-Edge representation . . . . . 25
Figure 21. Two cells with only a vertex in common . . . . . 27
Figure 22. A solid and its regularized complement . . . . . 31
Figure 23. Lexicographically ordering B-Rep$_{min}$ vertices and faces . . . . . 33
Figure 24. Canonically representing a bounding cycle of a face . . . . . 34
Figure 25. Canonically representing the boundary of a face . . . . . 34
Figure 26. Bottom of the staircase . . . . . 36
Figure 27. Tread of the staircase . . . . . 37
Figure 28. Riser of the staircase . . . . . 37
Figure 29. Sides of the staircase . . . . . 38
Figure 30. The completed staircase . . . . . 38
Figure 31. A solid with many more edges and vertices than faces . . . . . 39
Figure 32. Standard computation of $A \cap^* B$ . . . . . 45
Figure 33. Steps (1) and (2) of the new intersection algorithm . . . . . 47
Figure 34. Step (4) of the new intersection algorithm . . . . . 48
Figure 35. Triangulating the subdivision of intersecting faces . . . . . 51
Figure 36. The cross-section $G_P$ . . . . . 52
Figure 37. Intersecting faces of solid $B$ with plane $P$ . . . . . 53

Figure 38. Orienting edges of $G_P$ . . . 55
Figure 39. Isolated vertices of $G_P$ are in **in**$B$, **on**$B$, or **out**$B$ subsets of $P$ . . . 56
Figure 40. Local analysis at intersection points . . . 61
Figure 41. Intersecting $f$ and $G_P$ . . . 63
Figure 42. Cycle Containment Test . . . 63
Figure 43. Adding faces of $A$ in the interior of $B$ . . . 64
Figure 44. Transferring edges of $A \bigcap^{*} B$ to faces of $B$ . . . 67
Figure 45. Vertices of $A \bigcap^{*} B$ transferred to faces of $B$ . . . 69
Figure 46. Transferring a vertex in a face of $A$ . . . 70
Figure 47. Transferring a vertex in an edge of $A$ . . . 72
Figure 48. Transferring a vertex that coincides with a vertex of $A$ . . . 74
Figure 49. A subdivided face may not be a connected region . . . 77
Figure 50. Outer and Inner cycles of a face . . . 78
Figure 51. Organizing vertex-incidence tests . . . 90
Figure 52. A cross-face edge . . . 96
Figure 53. The Star-Edge Data Structure . . . 108
Figure 54. Embedding a cartesian coordinate system on a plane . . . 110
Figure 55. Localizing a vector to a principal axis or quadrant . . . 111
Figure 56. Data for the compound of five cubes . . . 113
Figure 57. The compound of five cubes . . . 114
Figure 58. A face of the compound of five cubes . . . 114
Figure 59. Intersecting two unit cubes . . . 116
Figure 60. Topological consistency . . . 117
Figure 61. Failure of the unit-cube experiment . . . 118
Figure 62. Constructed tetrahedron . . . 119
Figure 63. Data used to construct heptahedron . . . 121
Figure 64. Intersected octahedra ($\varepsilon = 10^{-6}$; $\theta = 0.001$ degrees) . . . 122
Figure 65. Polyhedral approximation to a sphere . . . 124

# List of Algorithms


Algorithm 1. Defining a face of B-Rep$_{min}$ .... 17
Algorithm 2. Orienting a directed edge .... 20
Algorithm 3. Regularized complementation using Star-Edge encodings .... 31
Algorithm 4. Converting from Star-Edge to canonical B-Rep$_{min}$ Encoding .... 35
Algorithm 5. The new regularized intersection algorithm .... 49
Algorithm 6. Intersecting the faces of solid $B$ with oriented plane $P$ .... 52
Algorithm 7. Orienting edges of $G_P$ .... 54
Algorithm 8. Isolated vertices of $G_P$ are in **in**$B$, **on**$B$, or **out**$B$ subsets of $P$ .... 56
Algorithm 9. Regularized intersection of $f$ and $G_P$ .... 58
Algorithm 10. Intersecting the edges and vertices of $f$ and $G_P$ .... 58
Algorithm 11. Local analysis at intersection points .... 61
Algorithm 12. Cycle Containment Test .... 63
Algorithm 13. Transferring edges of $A \cap^* B$ to faces of $B$ .... 66
Algorithm 14. Transferring a vertex in a face of $A$ .... 69
Algorithm 15. Transferring a vertex in an edge of $A$ .... 71
Algorithm 16. Transferring a vertex that coincides with a vertex of $A$ .... 73
Algorithm 17. Finding connected subsets of a subdivided face using a union-find program .. 77
Algorithm 18. Finding connected subsets of a subdivided face using point-location .... 79
Algorithm 19. Shell Containment Test .... 81
Algorithm 20. Finding the connected components of $A \cap^* B$ .... 82
Algorithm 21. Intersecting two single-shelled solids .... 83
Algorithm 22. Testing for deemed incidence .... 86
Algorithm 23. Testing for vertex/plane incidence .... 91
Algorithm 24. Testing for vertex/line incidence .... 91
Algorithm 25. Testing for vertex/ray incidence .... 92
Algorithm 26. Testing for edge/plane incidence .... 93
Algorithm 27. Simple cases in testing for edge/line incidence .... 93
Algorithm 28. The general case for edge/line incidence .... 94
Algorithm 29. Testing for edge/ray incidence .... 95
Algorithm 30. Testing for collinear-edge intersection .... 96
Algorithm 31. Testing for cross-face-edge/line incidence .... 97
Algorithm 32. Testing for collinear cross-face-edge/edge intersection .... 98
Algorithm 33. The general case for cross-face-edge/ray incidence .... 98
Algorithm 34. Radially ordering coplanar unit vectors .... 110
Algorithm 35. Radially ordering coplanar vectors using rational arithmetic .... 111
Algorithm 36. Inducing failure in an intersection algorithm .... 118

# List of Tables

Table 1. Timing data for the compound of five cubes ........................... 116
Table 2. Results of random-tetrahedron intersection-experiment .................... 121
Table 3. Timing data for sphere approximation ............................... 126

# Chapter 1. Introduction

Fundamental to computer-aided manufacture is the ability to design physical objects. One should be able to construct computer representations that describe these objects. Just as physical processes act on physical objects, computational analogues to these processes act on representations of the objects. For example, objects might be represented by their boundaries and set intersection might be an operation that we want to perform. Thus we need an algorithm that takes objects represented by their boundaries, and produces a boundary representation of their intersection. Algorithms that compute this intersection are difficult to implement reliably because implementations of such algorithms usually use finite-precision arithmetic and can fail or produce erroneous results. This thesis describes a new intersection algorithm that uses boundary representations of a certain class of physical objects called "solids" and uses symbolic inference to limit errors due to finite-precision arithmetic.

## *1.1 Defining Solids*

In this thesis, the term "solid" denotes any physical object whose surface consists of planar pieces. (See Figure 1 on page 2 for an example.) In order to precisely define a "solid," we first define the terms regular set, the regularization of a set, the regularized set operations, and regular convex polyhedra. The *regularization* of a set $A$, denoted $Reg(A)$, is defined as the closure of the interior of $A$, and a set $S$ is defined to be a *regular set* iff $S = Reg(S)$. The regularization of a set deletes every point from the boundary of that set with neighbourhoods that do not intersect the set interior (see Figure 2 on page 2). Just as there are set operations for manipulating sets, there are *regularized set operations*, $\bigcap^*$, $\bigcup^*$, $-^*$, and $\neg^*$, for manipulating regular sets (see Figure 3 on page 3 for an example).[1] Given sets $A$ and $B$, the regularized set operations are:

[1] Note that these operations are not restricted to regular sets, although we will usually apply them only to regular sets.

(1) $A \cap^* B \equiv Reg(A \cap B)$.
(2) $A \cup^* B \equiv Reg(A \cup B)$.
(3) $A -^* B \equiv Reg(A - B)$.
(4) $\neg^* A \equiv Reg(\neg A)$.

A *regular convex polytope* in $\mathbf{R}^3$ is a bounded point set formed as the regularized intersection of a finite number of closed half spaces. The *solids* are the finite-closure of regular convex polytopes in $\mathbf{R}^3$ under the regularized set operations. Thus a solid is the result of combining a bounded number of regular convex polytopes in $\mathbf{R}^3$ using regularized set operations.

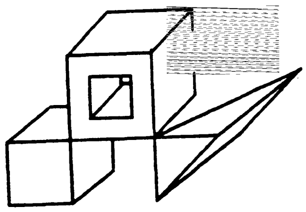
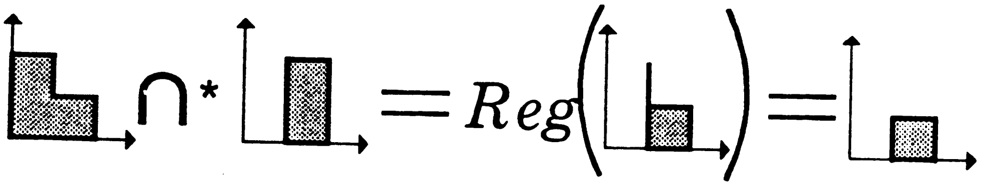


**Figure 1. A solid has planar surfaces**


In order to regularize $A$, boundary points which do not touch its interior are deleted.

**Figure 2. Regularization of a point set in $\mathbf{R}^2$**



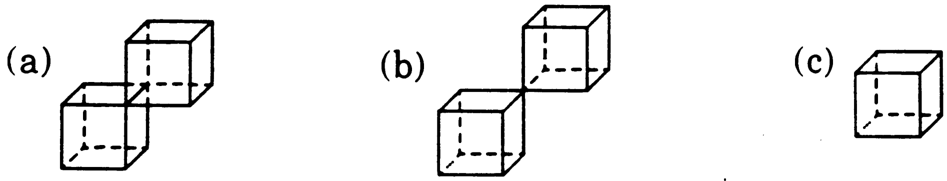

The regularized intersection of two sets is defined as the regularization of their intersection.

**Figure 3. Regularized Intersection**

## *1.2 Why study the intersection of solids?*

Because solids are closed under the regularized set operations they are an appropriate domain for studying intersection, although other domains have been used. For example, if we restrict the surface of a solid to be a 2-manifold, then we obtain objects called *manifold solids* (see Figure 4).

Although (non-manifold) solids constitute a larger class of objects than manifold solids, it has been argued that non-manifold solids are not practically useful, *i.e.*, not manufacturable [Eastman and Preiss, 1984; Mantyla, 1984]. In contrast, Requicha and Voelcker [1977] have argued that there are manufacturing processes such as material removal that can be described using non-manifold solids but not using manifold solids. It is still open to debate as to which of the two models is better, and the solid-modeling application itself dictates the class of physical objects modeled.




Solids (a,b) are not manifold solids because their surfaces are not 2-manifolds. Solid (c) is a manifold solid.

**Figure 4. Manifold solids are a proper subset of solids**

Rather than study an operation on a general domain such as solids, it is often easier to study the operation on a simpler domain comprising collections of objects simpler than solids. An *n-cell* is a topological space whose interior is homeomorphic to $\mathbf{R}^n$ (see Appendix A or Rourke and Sanderson [1982]), and a (*closed*) *cell complex* is a collection of cells with the following properties: (i) cells of the complex have pairwise-disjoint interiors; (ii) every cell of the complex has boundary that is the union of cells of the complex; and (iii) if two cells of the complex intersect, then their intersection is the union of cells of the complex. A 3-*complex* is a closed cell-complex with 0-, 1-, 2-, and 3-cells (see Figure 5). Dobkin and Laszlo [1987] described both a representation for certain 3-complexes and implementations of algorithms that use the representation. In particular, they discussed how to triangulate a polyhedron and construct a Delaunay triangulation of a point-set. Edelsbrunner, O'Rourke, and Seidel [1986] described algorithms that construct higher-dimensional Voronoi diagrams using a representation of cell-complexes due to Grunbaum [1967].

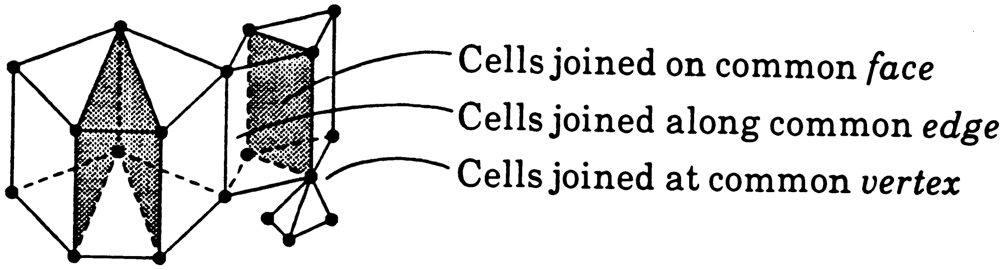



**Figure 5. A cell complex in $\mathbf{R}^3$**

Just as we argued that manifold solids are unsuitable for studying intersection because they are not closed under the regularized set operations, we can argue that 3-complexes are not closed under the set operations. It is of course possible to define an operation similar to regularization that converts the intersection of two 3-complexes into a 3-complex, for example, by triangulation. Such an operation would also convert a solid to a 3-complex. Conversely, a 3-complex can be converted into a solid by regularizing the union of its cells. The need to regularize demonstrates that there are 3-complexes that are not regular sets: Set $A$ of Figure 2 on page 2 is a cell-complex, but not a regular set. Thus we see that although cell-complexes are useful domains for the study of subdivision algorithms, solids seem to be more suitable for the study of set operations.

## *1.3 Representing Solids*

In order to formulate algorithms that manipulate solids we need a description, or representation, of these solids. (By "representation," we mean either an encoding scheme used to describe a solid or an *instance* of the scheme applied a particular solid.) Three types of representations from Requicha's [1980] taxonomy of representations are spatial decomposition, Constructive Solid Geometry, and boundary representation. The first of these, spatial decomposition, represents a solid as a disjoint union of simpler solids. The second, Constructive Solid Geometry, represents solids by explicitly giving a set-theoretic combination of geometric primitives, which parallels our definition of a solid. The third kind, boundary representation, represents the boundary of a solid as a collection of zero-, one-, and two-dimensional sets.

Spatial decomposition of a solid is a partition of its volume into simpler objects. For example, an octtree representation of a volume is a decomposition into cubes of volume $8^i v$, where $v$ is a constant. Thus an octtree approximates a volume with an error that depends on $v$. Octtrees tend to be large. For example, if the size of an octtree is the number of cubes in the representation, then an octtree can be exponentially larger than a corresponding Constructive Solid Geometry encoding (see Figure 6). Thus octtrees are not often used as the basis of a solid modeling system [Requicha, 1980], although spatial decomposition is generally used to represent complex mechanical-assemblies as parts [Requicha and Voelcker, 1982].

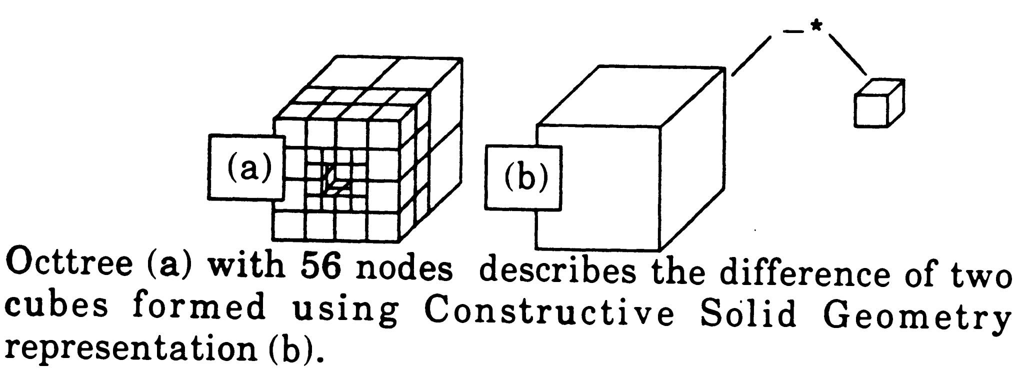
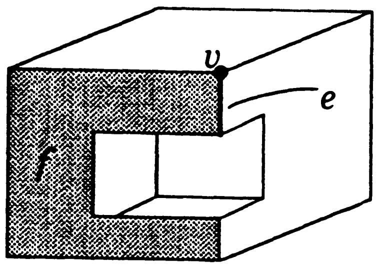



**Figure 6. Representations vary widely in size**

Constructive Solid Geometry (CSG) represents a solid by giving an algebraic expression where regularized set operators combine other solids. CSG representations are useful because related objects will often have similar representations. For example, the regularized complement of an object with CSG representation ($E$) has CSG representation $(\neg^{*}E)$ or $(\mathbf{R}^3 -^{*}E)$. However, not all solid modeling operations are easily implemented using CSG paradigms. For example, Lee and Requicha [1982a, 1982b] discuss the difficulties of using CSG in algorithms that compute volumes or moments of inertia.

A boundary representation ($B - Rep$) is an organized enumeration of certain zero-, one-, and two-dimensional sets of the surface of a solid which are called *vertices*, *edges*, and *faces*, respectively. We will see that precise definitions of these sets depend upon a particular boundary representation, although we will use these terms informally (see Figure 7).



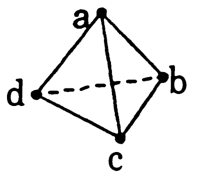

A *face* ($f$) is a connected, two-dimensional set on the surface of a solid.

A *vertex* ($v$) is the zero-dimensional intersection of three or more faces.

An *edge* ($e$) is a connected, one-dimensional set that is a component of the intersection of two or more faces.

**Figure 7. Vertex, edge, and face**

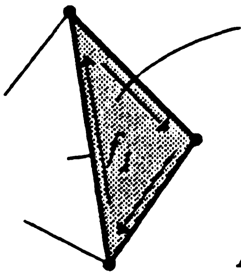
Boundary representations consist of two parts: the geometric information about the features (vertices, edges, and faces) of the boundary and the adjacency information about how these features are joined. For example, the geometric information in a B-Rep of a tetrahedron (see Figure 8 on page 7) might include the coordinates of the four vertices, and the adjacency information might consist of a clockwise listing of the vertices around each face. We will see that B-Reps can be more complicated than this simple example.




Tetrahedron

$a$: $(x_0,y_0,z_0)$
$b$: $(x_1,y_1,z_1)$
$c$: $(x_2,y_2,z_2)$
$d$: $(x_3,y_3,z_3)$

Geometric Information




$f_1$: $(c,a,b)$
$f_2$: $(d,a,c)$
$f_3$: $(d,c,b)$
$f_4$: $(a,d,b)$

Adjacency Information

Geometric information in the representation consists of the vertex coordinates. Adjacency information in the representation consists of an ordered list of vertices around each face.

**Figure 8. A boundary representation of a tetrahedron**

Boundary representations are often used to implement Constructive Solid Geometry. If a subexpression in a CSG representation is $E_1 \cap^* E_2$, then a regularized intersection algorithm computes a B-Rep that describes the object defined by $E_1 \cap^* E_2$: in this way a CSG representation is used as a "recipe" for a solid [Requicha and Voelcker, 1985], and computationally intensive operations, *e.g.*, finite-element analysis, use the B-Rep [Boyse and Rosen, 1982]. Thus B-Reps and algorithms for the regularized set operations are fundamental to solid modeling. We will now discuss algorithms for these operations.

## *1.4 Intersecting Solids Represented by their Boundaries*

Algorithms that intersect solids represented by their boundaries have been implemented for various classes of solids with varying degrees of completeness. These algorithms can be simplified by restricting the class of solids intersected or the way in which the solids are allowed to intersect. For example, if the solids are restricted to be convex polytopes, or if the surfaces of the solids are only allowed to intersect in a *non-degenerate* manner,[2] the intersection is easier to compute. Preparata and Shamos [1985] survey several algorithms that require time

[2] For example, given solids $A$ and $B$, a two-dimensional set, or face, of the surface of $A$ cannot intersect any face of $B$ in a two-dimensional set.

O($n \log n$) to compute the intersection of two three-dimensional convex polytopes (where $n$ is the number of vertices in the input convex polytopes), and Hertel, Mehlhorn, Mantyla, and Nievergelt [1985] describe O($n \log n$) algorithms that compute this intersection, as well as the associated union and difference. Intersecting solids is a more complicated problem than intersecting convex polytopes.[3] Allowing degenerate coincidences in the solid boundaries further complicates the problem. Tilove [1981] has pointed out that intersection algorithms must handle degenerate coincidences because they occur frequently in many solid-modeling applications. In addition, the intersection algorithms for convex polytopes, referenced above, all use the convexity property of the input polytopes in order to compute the intersection efficiently. However, it is doubtful that the techniques used to intersect convex polytopes are extensible to non-convex solids or manifold solids.

Algorithms for intersecting solids have been presented, although not proved correct, by several researchers. Laidlaw, Trumbore, and Hughes [1986] have described an algorithm for computing the regularized intersection of two solids, Mantyla [1986] has described an algorithm which computes the intersection of two manifold-solids, and Paoluzzi, Ramella, and Santarelli [1986] have described an algorithm for computing the regularized union of two solids. However, all of these algorithms have inherent shortcomings. The algorithm of Laidlaw, Trumbore and Hughes, although elegant, restricted faces of the representation to convex polygons, thereby complicating the algorithm in order to enforce this condition. Mantyla restricted his domain to manifold solids, thereby precluding the computation of a compound expression like $A \bigcap^{*}(B \bigcap^{*}C)$ because the subexpression $(B \bigcap^{*}C)$ is not necessarily a manifold solid. Paoluzzi, Ramella, and Santarelli used techniques similar to Mantyla's, restricted faces of their representation to be triangles, and claimed their algorithm worked for solids, as opposed to manifold solids.

## *1.5 Numerical Robustness of Geometric Algorithms*

One of the problems of implementing geometric algorithms is that computers use finite-precision arithmetic. Numerical quantities are represented as fixed-length bit-strings, and it is well known that not all fractions can be accurately described by computers in this way. Therefore, algorithms that make logical decisions

[3] Even from a theoretical standpoint, the problem of intersecting two convex polyhedra has a lower bound of $\Omega(n)$. The more general problem has an $\Omega(n^2)$ lower bound.

based upon finite-precision approximations are often prone to errors. Consider Figure 9 on page 9, in which the incidence of line-segment $e$ and line $P \cap Q$ is computed: endpoint $v$ of $e$ is on $P$, and endpoint $u$ is below $P$. An incidence test should not claim that a point in the interior of $e$ is incident to $P \cap Q$, but using a finite-precision approximation, it is possible that $A$ is incident to $Q$, and hence $e$ intersects $P \cap Q$. Algorithms which assume that $a$ and $v$, but not $u$, lie on $P$ are bound to fail: for example, a logical consequence of this assumption is that $u$, $a$, and $v$ define a plane.

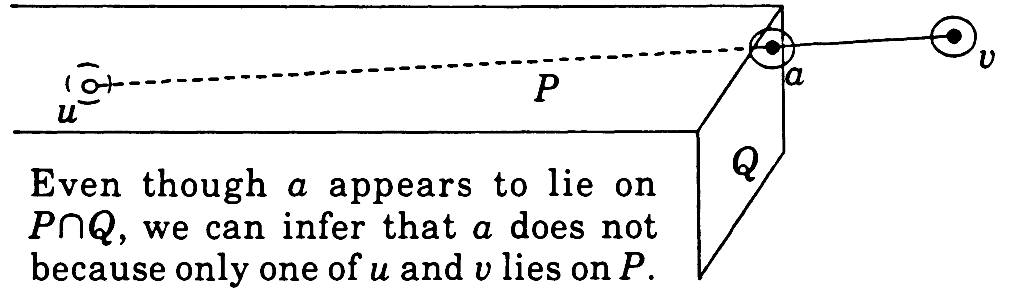



**Figure 9. Inconsistent finite-precision geometric computations**

Obviously, researchers need to seek *numerically-robust* algorithms, *i.e.*, algorithms that are provably correct using a model of finite-precision arithmetic, or at least algorithms that use finite-precision arithmetic and are empirically reliable. Dobkin and Silver [1988] describe some experimentation to show how numerical accuracy decreases as finite-precision calculations are composed. They also suggest that by iteratively perturbing input data and then repeating calculations, empirically good estimates of the precision of a computed quantity can be obtained. Hoffmann [1988] discusses their work and describes other several other approaches (some of which are also described in this section) to dealing with the problem of robustness in geometric algorithms.

Although there are as yet no provably correct algorithms that intersect solids using a model of finite-precision arithmetic, some work has been done on formulating robust algorithms for simpler geometric problems, such as computing the intersection of a collection of lines on a plane. (It may not be possible to exactly compute an intersection point even though the equations of two intersecting lines can be exactly represented.) There are two popular methods for achieving numerical robustness: (i) configurations which are inexact using a finite-precision approximation with a certain granularity are perturbed, maintaining topological invariants, into exact configurations; or (ii) sometimes algorithms can be formulated to work with inexact representations. Both of these techniques have been used in numerically-robust algorithms to intersect a collection of lines on a plane, producing sets of intersection points ordered along each line.

Algorithms have been given by Greene and Yao [1986] and Milenkovic [1986] that achieve numerical robustness by moving intersection points so that they are exactly representable using finite-precision. Greene and Yao used this technique to solve the line intersection problem, whereas Milenkovic used this technique to represent polygonal regions on a plane. In both cases, line segments were subdivided into smaller, non-collinear segments, thereby substantially increasing the complexity of the arrangements.

Algorithms formulated to work with inexact representations have been presented by both Milenkovic [1986] and Ottmann, Thiemt, and Ullrich [1987]. Ottmann, Thiemt, and Ullrich solved the line-segment intersection problem by computing intersection points using finite-precision arithmetic accurate to the least-significant bit.[4] The importance of their approach is that they were able to integrate this computation into the segment-intersection algorithm of Bentley and Ottmann [1979], without substantially changing it, and thereby making it numerically robust. Milenkovic modeled the error due to finite-precision by treating the lines as "wavy" curves. Because there can be many possible interpretations of curve-intersections, a data structure was produced which contains all possible interpretations. Milenkovic [1988] showed how to extract a desired interpretation from the data structure, using either an exponential-time algorithm or nonfinite-precision arithmetic.[5] Milenkovic has also shown how to intersect a set of planes in $\mathbf{R}^3$ by projecting intersection lines onto a plane, using a robust line-intersection algorithm, and then back-projecting.

Greene and Yao [1986], Milenkovic [1986, 1988], and Ottmann, Thiemt, and Ullrich [1987] solved one problem, computing the intersection of a collection of coplanar lines, that arises in many geometric algorithms; however, many other problems must be addressed in order to compute the regularized intersection of two solids. (see, *e.g.*, Figure 9 on page 9). Segal and Séquin [1985] suggested

---

[4] If a fraction is represented using a string of $k$ bits, then a computation is performed with an error of at most $2^{-k}$.

[5] If the equations of the lines to be intersected are represented using $k$ bits of precision, then Milenkovic's algorithm requires $k(1 + c)$ bits of precision, where $c$ is a small constant. (For $k \leq 64$, $k(1 + c) \approx 11$.)

how a boundary representation for a solid might be changed by perturbing vertices to points on a discrete grid, and then by modifying the other parts of the representation. Although relevant, their work only begins to address the issue of numerical robustness. Sugihara [1987] described a technique in which certain parts of the geometric information in a boundary representation are represented exactly, and other parts are approximated using finite precision.

We have seen that boundary representations and regularized set operations are fundamental to solid modeling and that regularized intersection algorithms are complicated. Complications arise from the need to handle, in a numerically robust and efficient way, degeneracies in the relative positions of the two solids. In this thesis, an algorithm to construct the regularized intersection of two solids represented by their boundaries is presented and proved correct using a model of exact arithmetic. The algorithm is implemented to use symbolic inference in order to limit the errors from finite-precision arithmetic in incidence computations, and experimental evidence is presented that demonstrates a measure of numerical robustness. The intersection algorithm uses a new boundary representation, called the *Star-Edge* representation, defined in "Chapter 2. Representing Solids".

# Chapter 2. Representing Solids

The regularized intersection algorithm studied in this thesis uses a new boundary representation for solids, called the Star-Edge representation. To show why this new boundary representation is needed, some representational issues are discussed. Then the Star-Edge representation is defined and compared to other, existing representations.

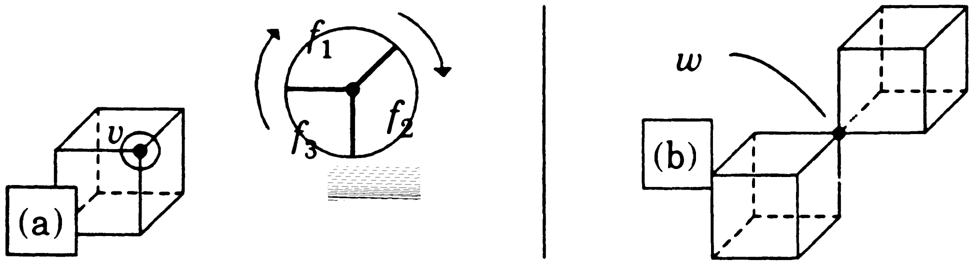
Recall from "Chapter 1. Introduction" that a boundary representation (B-Rep) is an enumeration of zero-, one-, and two-dimensional sets, called vertices, edges, and faces, respectively, that constitute the boundary of a solid. Exact definitions of these sets are necessary to define a boundary representation [Brown, 1981]. However, there are two fundamental assumptions about the faces of a boundary representation that simplify the formulation of algorithms that use boundary representations. These assumptions are: (1) all faces in a B-Rep are regular sets with pairwise-disjoint interiors;[6] and (2) the union of all faces in a B-Rep covers the total surface of the solid. A more general issue is whether a regularized intersection algorithm should use boundary representations for manifold or non-manifold solids.

Certainly boundary representations for manifold solids can be simpler than for solids because representations for 2-manifolds can be used to describe manifold solids [Baumgart, 1972; Guibas and Stolfi, 1985]; an edge in the boundary representation of a manifold solid is defined by the intersection of two surfaces; a vertex is defined by the intersection of three or more surfaces; and the surfaces whose intersection defines a vertex can be radially ordered around that vertex. In contrast, the surfaces whose intersection defines a vertex of a non-manifold solid cannot be ordered as easily (see Figure 10 on page 13). Because manifold solids are easier to represent than non-manifold solids, there have been attempts to use boundary representations for manifold solids in regularized intersection

---

[6] In order for this property to hold, we must be careful how we define the "boundary" and "interior" of a face.

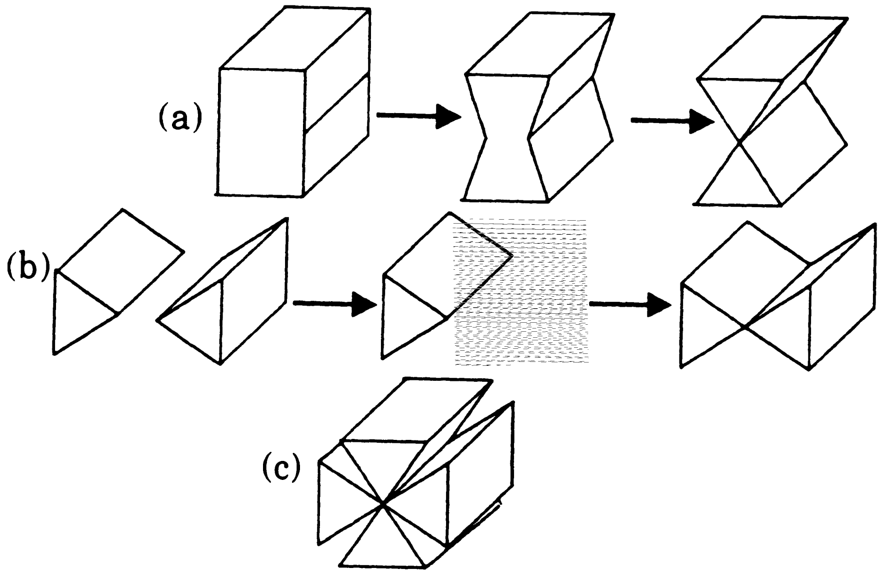
algorithms, even though manifold solids are not closed under regularized set operations.




Because the surface of manifold solid (a) is a 2-manifold, we can represent vertex $v$ by the radially-ordered face list $(f_1, f_2, f_3)$. There is no radial ordering of the six faces incident to vertex $w$ in solid (b).

**Figure 10. Manifold-solid vertices have simple neighbourhoods**

Even though non-manifold solids have more complicated boundaries than manifold solids, Weiler [1984] demonstrated that by moving and deforming a manifold solid, it can be made to look like a non-manifold solid. The technique is illustrated in Figure 11(a), where a 2-manifold (the surface of a cube) is deformed into an "hourglass." Conceptually, this deformation works by constructing a 2-manifold B-Rep for the cube and modifying the geometric part of the B-Rep (see "Chapter 1. Introduction") to look like an hourglass. Thus, Weiler treats the hourglass as a deformed cube, even though the embedding of the hourglass in $\mathbf{R}^3$ is not bounded by a 2-manifold. Weiler's technique has been criticized by Eastman and Preiss [1984] and Mantyla [1984] because it is difficult to manipulate such boundary representations. For example, the hourglass of Figure 11(a) can be constructed by moving two prisms together as shown in Figure 11(b), and positional uncertainties in the two coincident line segments can lead to numerical errors in algorithms that use a manifold boundary representation for this hourglass. This possibility of error arises because the fact that two line segments are coincident is not recorded in the representation. Even in the absence of positional uncertainty, such representations are difficult to combine into representations of new objects. For example, a manifold boundary representation of the object shown in Figure 11(c) is difficult to construct from manifold boundary representations of the objects in Figure 11(a,b). Despite the difficulties described here in approximating the boundary of a non-manifold solid using a 2-manifold, some researchers have formulated algorithms for regularized set operations which use this (Weiler's) technique. Mantyla [1986] was able to use Weiler's technique to intersect manifold solids, but did not discuss how to intersect solids. Paoluzzi, Ramella, and Santarelli [1986] presented a union algorithm for solids that used Weiler's technique on a B-Rep with triangular faces.

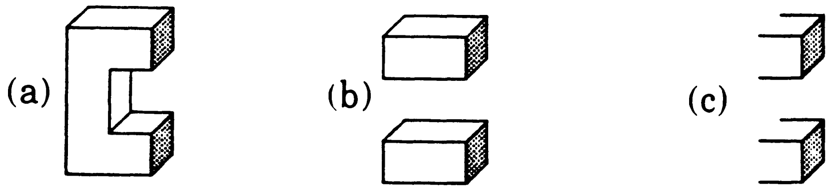



In (a) we form an "hourglass" by deforming a cube. In (b) we form an hourglass by moving two prisms together. In (c) the union of these two hourglasses has an edge that is adjacent to eight faces.

**Figure 11. Weiler's technique for representing solids**

We conclude from the above discussion about Weiler's technique that a boundary representation to be used for a class of solids should completely describe incidences in the solid's surface. This is not the case, for example, if we use a representation for a 2-manifold to describe the surfaces of the objects in Figure 11. We define a boundary representation as *explicit* for a class of solids if, using appropriate definitions for features (vertices, edges and faces), that boundary representation describes all the incidences of the features of any represented solid in the class. For example, the boundary representations of Stolfi and Guibas [1985] and Baumgart [1972] are explicit for manifold solids; they define a face as a polygon (and its interior); an edge is a line segment that is a component of the intersection of two faces; a vertex is the intersection of three or more faces. We will see that the Star-Edge representation of this thesis is explicit for solids.

Even though precise definition of a face depends on a particular boundary representation, algorithms that manipulate B-Reps are easier to formulate if faces of the B-Rep are connected sets. For example, the fact that *faces* in a boundary representation are connected sets makes it easier to determine if the represented *solid* is a connected set (see Figure 12). Thus we define a boundary representation to be a *connected* boundary representation if the faces of the representation are restricted to be connected sets.

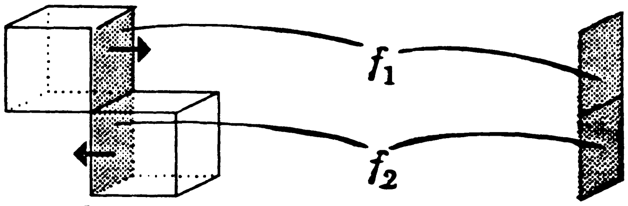



The solid in (a) is a connected set; the solid in (b) is not. Because a representation that is not connected does not discriminate between these two situations, (c) cannot immediately be classified as connected or not.

**Figure 12. Face and solid connectivity**

It is important for many geometric algorithms to have outward normals associated with the faces of a boundary representation of a solid. Outward normals are used to discriminate between points in the interior and exterior of the solid. For example, outward normals can be used to determine if a point is in the interior of a convex polyhedron. Assigning an outward normal to a face puts a requirement on the neighbourhood of the points of a face. A face $f$ of a boundary representation of solid $S$ is defined to have the *constant-neighbourhood property* [Requicha, 1980] if there is a plane $P(x,y,z)$ defined by $Ax + By + Cz + D = 0$, containing $f$, such that for half-space $H$ defined by $\{(x,y,z) : P(x,y,z) \leq 0\}$, every neighbourhood $N$ of every point $a$ of $f$ has the property that $N \cap H \cap Int(S)$ is non-empty. We will require that boundary representations have faces with the constant neighbourhood property (see Figure 13 on page 16).

**Notation**: From the above definition, plane $P(x,y,z)$ is called the *oriented bounding-plane* of face $f$. For brevity, we say that $P$ is the *plane* of $f$, and the oriented bounding-planes of the faces of a solid are called the *planes of the solid*.




$F = f_1 \cup f_2$
$Int(f_1) \cap Int(f_2) = \varnothing$

Regular set $F$ does not have the constant neighbourhood property because $f_1$ and $f_2$ have different outward normals, but regular sets $f_1$ and $f_2$ do have the constant neighbourhood property.

Figure 13. The neighbourhood of a point of a face

The discussion to this point (in this chapter) abstracts from previous research. (see also [Requicha, 1980; Brown, 1981; Silva, 1981; Eastman and Preiss, 1984; Mantyla, 1984].) From this discussion we extract five properties of boundary representations which seem to allow algorithms that use boundary representations to be easier to formulate (see Figure 14). The new boundary representation of this thesis, called the Star-Edge representation, has these properties.

(1) faces in a B-Rep are regular sets with pairwise-disjoint interiors;

(2) faces in a B-Rep cover the surface of the represented solid;

(3) boundary representations are explicit (for solids);

(4) boundary representations are connected; and

(5) faces in a B-Rep have a constant neighbourhood.

Figure 14. Representational properties

## 2.1 A Minimal Boundary Representation for Solids

In order to define the Star-Edge representation for solids, it is necessary to define the features (vertices, edges, and faces) of the representation. We will do this by defining the features of a very general B-Rep for solids, and then restrict these definitions appropriately. We call this general boundary representation a *minimal* *boundary representation*, denoted B-Rep$_{min}$. We will define what we mean by a face of B-Rep$_{min}$ first, and then define vertices and edges as intersections of these faces.

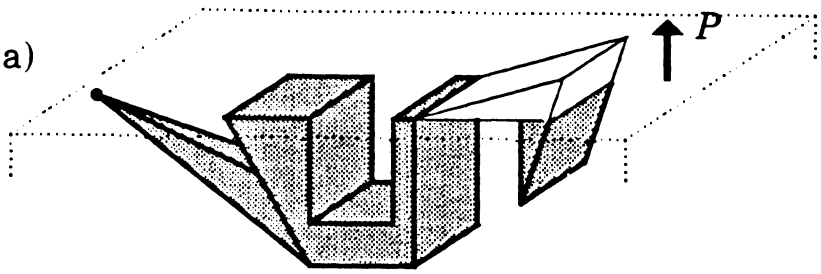
In the previous section we said that the faces of a B-Rep are regular sets with pairwise-disjoint interiors that cover the surface of the solid, and whose interior points have a constant neighbourhood. A procedural definition of a face is given as Algorithm 1. Once a face is defined, we can define what we mean by the face interior and boundary in order that Properties (1) and (3) of the representational properties (Figure 14 on page 16) hold (face interiors are disjoint and all incidences are enumerated).

Let $P$ be a plane of solid $X$, and let $H$ be the closed half-space bounded by $P$ such that the normal vector to $P$ points away from $H$. The B-Rep$_{min}$ face on $P$ is defined as follows:

(1) Compute the regularized intersection of $H$ with the interior of $X$; denote the result by $X_H$ (see Figure 15[a]).
(2) Compute the intersection of $X_H$ with $P$, and regularize the result with respect to $P$; the result, denoted $X_P$, is a regular point set on $P$ that represents the cross section of $X$ w.r.t. $P$ (see Figure 15[b]).
(3) Compute the regularized difference of $X_P$ and the interior of $X$; the result is a B-Rep$_{min}$ face of $X$ on $P$ (see Figure 15[c]).

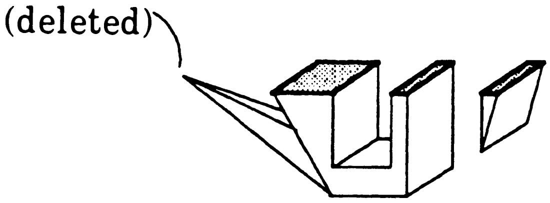
**Algorithm 1. Defining a face of B-Rep$_{min}$**

(a)



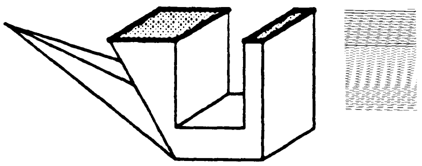

Regularized intersection of interior of solid $X$ with half-space $H$ is denoted $X_H$.

(b)




Isolate points on $P$ by intersecting $X_H$ with $P$ and the regularizing with respect to $P$; denote the result by $X_P$.

(c)



Remove points from $X_P$ that are in the interior of $X$ by computing the regularized difference of $X_P$ and the interior of $X$. (Both the rectangle and its interior disappear after regularizing.)

**Figure 15. Defining a face of B-Rep$_{\min}$**

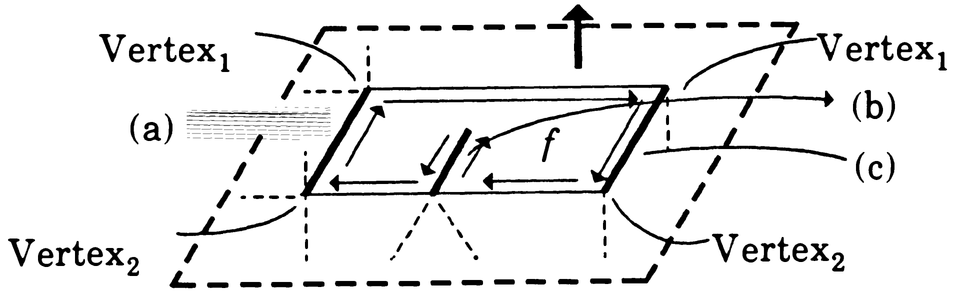
With the definition of a face given in Algorithm 1 on page 17, we can define what we mean by the boundary and interior of a face, and the edges and vertices of B-Rep$_{\min}$. First, the *boundary of a $B - Rep_{\min}$ face* is defined as the intersection of that face with all other B-Rep$_{\min}$ faces in the representation. Faces intersect in line segments and points, called B-Rep$_{\min}$ *edges* and *vertices*, respectively. A point is defined to be in the *interior of a $B - Rep_{\min}$ face* if that point is in the face and not in the face-boundary. An immediate consequence of this approach to defining B-Rep$_{\min}$ is that all of the representational properties (Figure 14 on page 16) are satisfied, with the exception of connectedness (Property [4]), as demonstrated by Figure 15(c). In the next section we define the Star-Edge representation; we refine the faces of B-Rep$_{min}$ into connected sets (satisfying Property [4]); and we define mechanisms, called *adjacency mechanisms*, for describing relationships between the vertices, edges, and faces of the Star-Edge representation.

## 2.2 The Star-Edge Boundary Representation

We now define the Star-Edge representation by refining definitions of the vertices, edges and faces of B-Rep$_{min}$. Boundary representations whose faces are triangles [Paoluzzi, Ramella, and Santarelli, 1986], polygons [Baumgart, 1972; Guibas and Stolfi, 1985], or convex polygons [Laidlaw, Trumbore, and Hughes, 1986] have been defined, but we choose a less restricted definition by requiring only that the representational properties (Figure 14 on page 16) hold. Consequently, Star-Edge faces can be as simple as the triangular faces of Paoluzzi, Ramella, and Santarelli, or as complicated as the connected components of the faces of B-Rep$_{min}$. With definitions of the vertices, edges, and faces of the Star-Edge representation, we present adjacency mechanisms that describe incidences. (An encoding of the Star-Edge representation used by an implementation of the regularized intersection algorithm of this thesis is described in Appendix B.)

The faces, edges, and vertices of the Star-Edge representation are now defined. Subdivide the B-Rep$_{min}$ faces of a solid into connected, regular sets. Each set in the subdivision is a *Star-Edge face.* The *boundary of a Star-Edge face* is defined as the intersection of that face with all other Star-Edge faces in the representation. Faces intersect in line segments and points, called Star-Edge *edges* and *vertices*, respectively. A point is defined to be in the *interior of a Star-Edge face* if that point is in the face but not the face-boundary. The definitions of Star-Edge faces, edges, and vertices closely resemble their B-Rep$_{min}$ analogues. A consequence of this is that all of the representational properties (Figure 14 on page 16) are satisfied. We now define the adjacency mechanisms of the Star-Edge representation.

An edge is a line segment in $\mathbf{R}^3$ bounded by two vertices; one of these vertices is arbitrarily called Vertex$_1$ and the other is called Vertex$_2$. Informally, an edge is oriented with respect to an incident face as follows: when traversing an edge of a face from Vertex$_1$ to Vertex$_2$, if the face is to the right of the edge then the edge is assigned a *right orientation* with respect to the face. If the face is to the left of the edge then the edge is assigned a *left orientation.* It is possible to have the face both to the right and to the left, and in this case the edge is assigned both a right and left orientation (see Figure 16 on page 20); this procedure is formalized as Algorithm 2 on page 20. We call these oriented face/edge pairs *directed edges* [Karasick, 1989]. The coincidence of an edge and a face is represented either by one or two directed edges, and in general, the directed edges associated with an edge describe the orientations of that edge on all incident faces in the Star-Edge representation.




Edge (a) gets a left orientation on $f$; edge (b) gets both a left and a right orientation on $f$; edge (c) gets a right orientation on $f$.

**Figure 16. Orienting a face/edge intersection**

An edge $e$ contained in face $f$ is assigned a left orientation, a right orientation, or both, by the following procedure. This procedure assumes the existence of the normal vector $N$ to $f$ which points to the exterior of the represented solid.

(1) First, construct a vector $t$, defined by $Vertex_2(e) - Vertex_1(e)$ and then compute the two vectors $l$ and $r$ in the plane of $f$ that point from the interior of $e$ to the left and to the right: $r$ is $t \times N$, and $l$ is $\leftarrow N \times t$.

(2) Either $l$ or $r$, or both, point from $e$ to the interior of $f$. If $l$ points into the interior of $f$, then $e$ gets a left orientation on $f$. If $r$ points into the interior of $f$, then $e$ gets a right orientation on $f$. If both $l$ and $r$ lie in the interior of $f$ then $e$ gets both a left and a right orientation (see Figure 16).

**Algorithm 2. Orienting a directed edge**

A directed edge ($\vec{e}$) has three components: an edge $e$ given by ($Vertex_1$, $Vertex_2$), a containing face $f$, and an orientation bit *right?* (which is true if the directed edge has a right orientation). These three data are used to infer numerical quantities such as the tangent $t(\vec{e})$ corresponding to the traversal direction of the edge, and the initial and terminal vertices of a directed edge. A vector, called a face direction vector, which points from the edge into the interior of an incident face, is inferred as well. Definitions for all of these quantities are found in Figure 17.

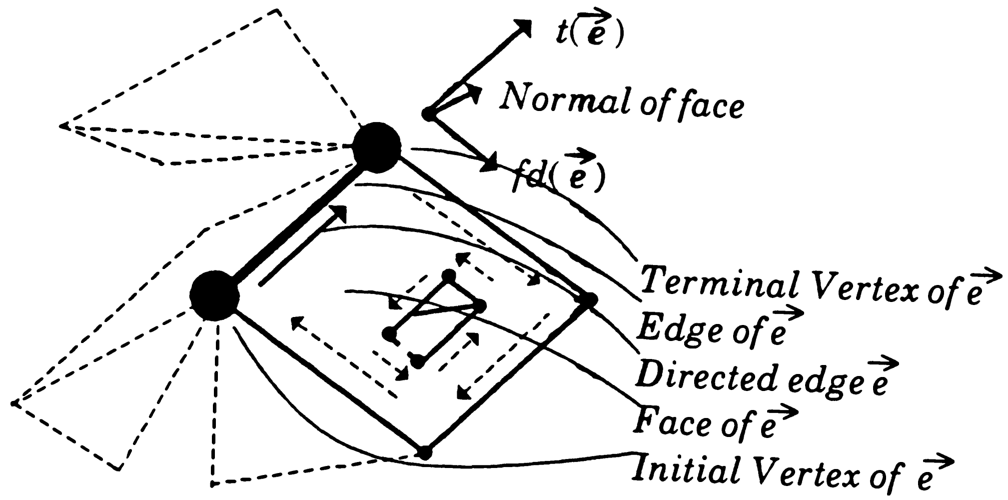



**Figure 17. Directed edge notation:** We denote directed edge $\vec{e}$ by the triple ($e$,$f$,*right?*), where $e$ is the edge of $\vec{e}$, $f$ is the face of $\vec{e}$, and *right?* is the orientation bit of $\vec{e}$. Note that there may be many directed edges associated with $e$. We say that $\vec{e}$ is one of the directed edges of $e$. Directed edge $\vec{e}$ describes how $e$ is traversed on $f$: if $\vec{e}$ has a right orientation, then we say that $Vertex_1(e)$ is its *initial vertex*, and $Vertex_2(e)$ is its *terminal vertex*; conversely, $\vec{e}$ has a left orientation, then we say that $Vertex_2(e)$ is its initial vertex, and $Vertex_1(e)$ is its terminal vertex. The *tangent* of $\vec{e}$, denoted $t(\vec{e})$, is defined as the unit vector directed from its initial vertex toward its terminal vertex. Finally, we define the *face direction* of $\vec{e}$, denoted $fd(\vec{e})$, as the unit vector that is the cross product of $t(\vec{e})$ with the normal vector to the face $f$ of $\vec{e}$. This vector, which is in the plane of $f$, points orthogonally from the interior of $e$ into the interior of $f$.

We can use the directed edge notation (Figure 17) to describe a representation for the directed edges of a Star-Edge face, which forms part of the face boundary.[7] First, the directed edges of a face are ordered radially around incident

[7] Recall that by *boundary of a face* we mean the intersection of that face with all other faces in the representation.

vertices on that face (see Appendix C). Using this ordering, the *bounding directed-edge cycles of a face* (denoted simply as "cycles") are defined as those directed-edge cycles induced using the following rule to determine succession: given a directed edge $\vec{e}_1$, the next directed edge $\vec{e}_2$ around the bounding directed-edge cycle is the counter-clockwise successor to $\vec{e}_1$ in the radial ordering of directed edges around the terminal vertex of $\vec{e}_1$ (see Figure 18).[8] We arbitrarily distinguish one directed edge as the first one in a cycle in order to facilitate traversal of this cycle. It is possible that vertices incident to a face are not contained in any of its cycles. We say that these vertices are *isolated vertices* of this face. Thus, we represent the boundary of a Star-Edge face as a list of isolated vertices and bounding directed-edge cycles.

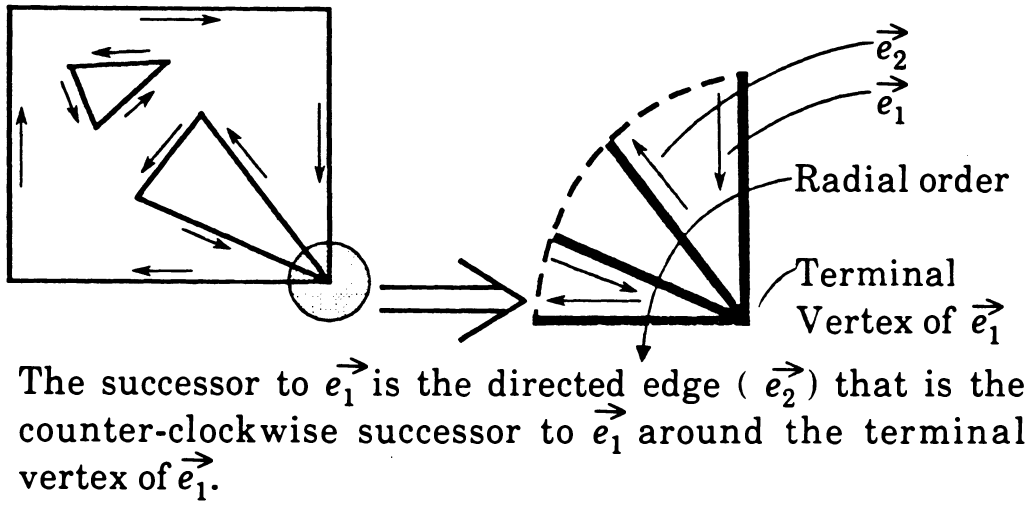



**Figure 18. The successor of a directed edge of a face**

## *2.3 Some Combinatorial Properties of the Star-Edge Representation*

Now that the Star-Edge representation has been defined, we can make some observations about properties of the representation. Because of our definition of a bounding directed-edge cycle, a vertex is alternately an initial and terminal vertex

[8] This rule induces a clockwise cycle, *i.e.*, the face is always found to the right of its directed edges; if instead we used the radially-clockwise successor at the terminal vertices to determine the next directed edge around the cycle, the face would be found the the left of its directed edges, and we would induce a counter-clockwise cycle.

in the radial ordering of the directed edges of a face incident to that vertex. Thus on each incident face of a vertex there are an even number of directed edges incident to that vertex, and a directed edge together with its successor directed edge "enclose" an area of this face. Similarly, an edge may be adjacent to many faces, and associated with each adjacent face is one or two directed edges. In total, an edge has an even number of directed edges, and the faces of these directed edges can be radially ordered around this edge by using face direction vectors to sort them (see Appendix C). Thus the faces of two consecutive directed edges in this ordering enclose a volume of the represented solid. These two radial orderings of directed edges around vertices and edges are shown in Figure 19.

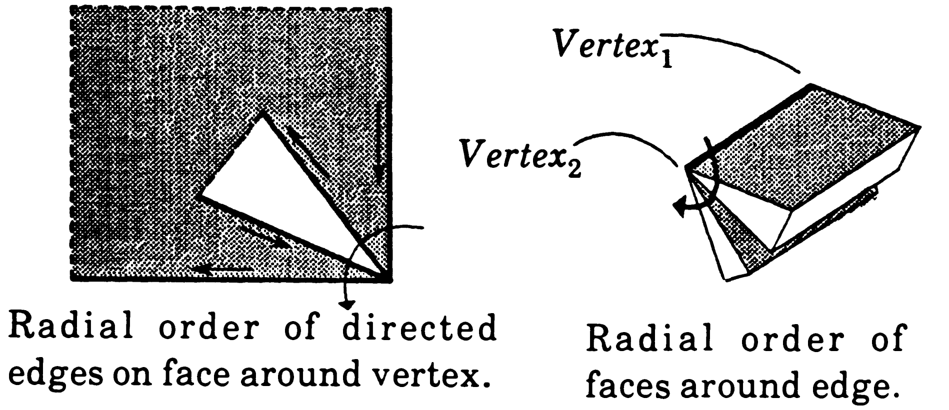



**Figure 19. Radial ordering of directed edges**

Asymptotic analyses of algorithms that use the Star-Edge representation of a solid $A$ are parameterized by the numbers of vertices, edges, directed edges and faces of the Star-Edge representation of $A$. We denote these quantities by $V_A$, $E_A$, $D_A$, and $F_A$, respectively, and we further denote quantities like "the number of directed edges adjacent to vertex $v$" by "$D_v$." We now $V_A$ and $F_A$ in terms of $E_A$.

**Lemma 1.** If a Star-Edge representation of solid $A$ has $V_A$ vertices, $F_A$ faces, and $E_A$ edges, and every vertex is adjacent to at least two edges and two faces, then: (1) $V_A \leq E_A$; and (2) $2F_A \leq 3E_A$.

**Proof.**

(1) Each vertex is adjacent to at least two edges, and so by listing the the edges adjacent to each vertex, each edge is listed twice:

$$2V_A \leq \sum_{v}^{Vertices(A)} E_v = 2E_A.$$

(2) Each vertex is adjacent to at least two faces, and so $2F_A \leq \sum_{v}^{Vertices(A)} F_v$. The number of faces $F_v$ adjacent to vertex $v$ is bounded by one more than the number of edges adjacent to the $v$. ($v$ can be isolated on exactly one adjacent faces.) Finally, using Result (1), we have

$$2F_A \leq \sum_{v}^{Vertices(A)} F_v \leq \sum_{v}^{Vertices(A)} (E_v + 1) \leq 3E_A \quad ■$$

There are also trivial relationships between the numbers of edges and directed edges in the Star-Edge representation of a solid.

**Lemma 2.** If a Star-Edge representation of solid $A$ has $D_A$ directed edges and $E_A$ edges, then: (1) $2E_A \leq D_A$; and (2) for every face $f$ of the representation, $2E_f \geq D_f$.

**Proof.** Every edge of the Star-Edge representation of $A$ is adjacent to at least two faces of the representation. ■

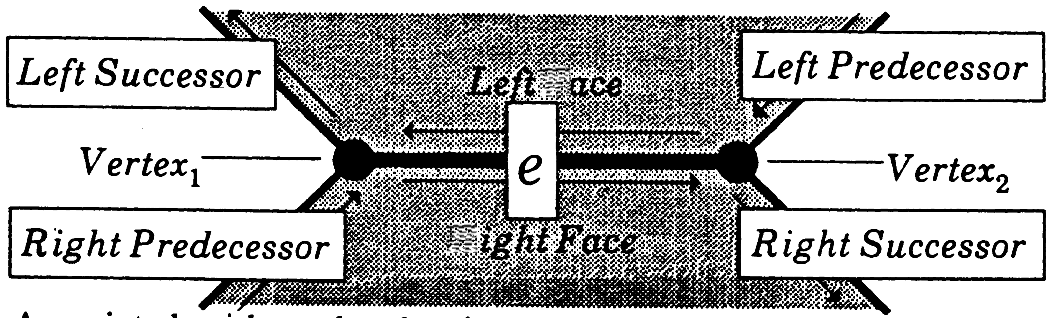
We will use these bounds to obtain a simple form for the asymptotic running time required by the intersection algorithm of "Chapter 4. Intersecting Solids".

## *2.4 A Survey of Boundary Representations*

This chapter began with a discussion of representational issues. A very general B-Rep, called B-Rep$_{min}$, was defined and used to define the Star-Edge representation. We now compare the Star-Edge representation to other, existing boundary representations for manifold-solids, non-manifold solids, and cell complexes. Manifold-solid representations discussed are the *Winged-Edge* boundary representation of Baumgart [1972], the *Quad-Edge* boundary representation of Stolfi and Guibas [1985], and the *Winged-Triangle* boundary representation of Paoluzzi, Ramella, and Santarelli [1986]. We will also examine representations for cell-complexes in $\mathbf{R}^3$ [Hanrahan 1985; Dobkin and Laszlo 1987] and a representation for solids due to Laidlaw, Trumbore, and Hughes [1986].

The Winged-Edge representation [Baumgart, 1972] and the Quad-Edge representation [Stolfi and Guibas, 1985] are explicit, connected representations for manifold solids. Property (4) of the representational properties (Figure 14 on page 16) is enforced by requiring that face boundaries be simple polygons, *i.e.*, that topologically, faces are discs. The Winged-Triangle representation

[Paoluzzi, Ramella, and Santarelli, 1986] further requires that face boundaries be triangles. The Winged-Edge, Quad-Edge, and Winged-Triangle representations for manifold solids exploit two properties of manifold solids: an edge in the representation of a manifold-solid is defined by the intersection of exactly two faces, and the edges incident to a vertex in the representation can be radially ordered around that vertex. Vertex coordinates serve as a means for embedding a 2-manifold in $\mathbf{R}^3$ as the surface of a manifold solid. Thus, a very simple adjacency mechanism suffices to describe the topology of a 2-manifold: this mechanism is a list of the preceeding and succeeding edges on the two adjacent faces of an edge (see Figure 20). The adjacency mechanism for the Winged-Triangle representation is even simpler because each triangle (face) is adjacent to three other triangles. It is easy to see, by using directed edge notation, how representations for manifold solids are special cases of the Star-Edge representation. If the Star-Edge representation is used to represent a manifold solid, then each edge has exactly two directed edges and each vertex of a face is adjacent to two directed edges on that face.

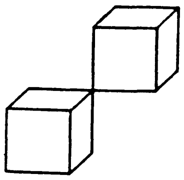



Associated with each edge in the Winged-Edge representation are the four edges that correspond to the predecessor and successor edges on the left and right faces. The successor (clockwise) to $e$ around a face is the same as the successor (counter-clockwise) to $e$ around a vertex.

**Figure 20. Adjacencies of the Winged-Edge representation**

Although polygonal face-boundaries are easier to represent than the general face-boundaries of the Star-Edge representation, it is necessary to examine how polygonal face-boundaries affect algorithms that use such representations. Polygonal or triangular face-boundaries increase the number of faces in a representation. This has two effects on regularized intersection algorithms: because face boundaries of these representations are polygonal, extra work in addition to the actual intersection computation is necessary; on the other hand, the intersection of two faces becomes easier to compute (but it is not clear if this simplicity outweighs the other induced computation [*c.f.*, Eastman and Preiss, 1984]). Certain algorithms require polygonal face-boundaries. For example, the Quad-Edge representation of Guibas and Stolfi [1985] can be used to describe subdivisions of $\mathbf{R}^2$, and was used in an algorithm to compute the Voronoi diagram and Delaunay triangulation of a set of points in $\mathbf{R}^2$.

Representations for cell-complexes embedded in $\mathbf{R}^3$ have been described by Hanrahan [1985] and Dobkin and Laszlo [1987]. Hanrahan's representation, the *Face-Edge* representation, was used in a solid modeling system. Dobkin and Laszlo independently formulated the *Facet-Edge* representation, which they used to solve geometric problems like the construction of Voronoi diagrams and Delaunay triangulations in $\mathbf{R}^3$. The Face-Edge and Facet-Edge representations, like the Star-Edge representation, are based on intersections of edges with faces (recall that Star-Edge nomenclature for the intersection of an edge with a face is a directed edge). Not surprisingly, the Star-Edge representation has adjacency mechanisms which are very similar to those used by these two representations. Dobkin and Laszlo supplied adjacency mechanisms for the Facet-Edge representation which they called traversal functions. These functions provide the four facet/edge traversals; using directed edge notation, these traversals correspond to moving around a face or an edge to the successor or predecessor of a directed edge. (For the Star-Edge representation, we defined only the successor of a directed edge around a face, but the other three traversal functions could easily have been defined; see Figure 19 on page 23.) One important difference between the Star-Edge representation and the Facet- and Face-Edge representations is that the Star-Edge representation can describe certain cell complexes which cannot be described using these representations (see Figure 21 on page 27).



This cell complex cannot be represented using either the Face- or Facet-Edge representations because these representations require that adjacent cells intersect in at least an edge and these cubes intersect at only a vertex.

**Figure 21. Two cells with only a vertex in common**

There is a representation for solids, due to Laidlaw, Trumbore, and Hughes [1986], that is closely related to the Star-Edge representation. In Laidlaw, Trumbore, and Hughes' representation, faces are bounded by convex polygons. This simplifies the adjacency mechanisms in two ways: edges are defined by an even number of faces, and vertices are defined by an even number of edge/face pairs (directed edges). In contrast, Star-Edge faces are much more general: an edge has an even number of directed edges, and on each face a vertex is adjacent to an even number of directed edges. Because Laidlaw, Trumbore, and Hughes bound faces with convex polygons, their representation may contain many more faces than the corresponding Star-Edge representation.[9] A regularized intersection algorithm that uses the Laidlaw, Trumbore, and Hughes representation is complicated by three factors: first, the input representations are quite lengthy; and second, extra complication is added to the intersection algorithm in order to ensure face convexity; and third, many edges in their representation are not defined by the intersection of two faces, and this can complicate incidence tests in their intersection algorithm. Laidlaw, Trumbore, and Hughes designed their representation in this way hoping that the ease with which two convex faces could be intersected would compensate for these drawbacks. On the other hand, the Star-Edge representation is designed to minimize the number of faces in order to lower the overall cost of these computations.

---

[9] Subdividing a single Star-Edge face with $n$ directed edges can induce as many as $n$ convex faces.

We have seen that the Star-Edge representation subsumes many other representations for solids and manifold solids. Certainly the Star-Edge representation is more general than the manifold-solid B-Reps of Baumgart [1972], Stolfi and Guibas [1985] and Paoluzzi, Ramella, and Santarelli [1986]. For manifold solids, instances of these three representations are easily converted into instances of the Star-Edge representation. Furthermore, we saw that there are cell-complexes which the Star-Edge representation can describe, and the representations of Hanrahan [1985] and Dobkin and Laszlo [1987] cannot. Finally, the boundary representation of Laidlaw, Trumbore, and Hughes [1986] for solids is easily converted into an instance of the Star-Edge representation. In the next chapter we examine how to solve two basic problems using the Star-Edge representation: (i) computing the regularized complement of a solid, and (ii) determining if two instances of the Star-Edge representation describe the same solid.

# Chapter 3. Some Representational Results

To demonstrate the Star-Edge representation's utility, we can show how to use the Star-Edge representation in algorithms for solid-modeling operations other than regularized set operations. The Star-Edge representation will now be used to solve two problems in solid modeling: (1) transform a representation of a solid into one for the solid's regularized complement; and (2) determine if two solids, represented by their boundaries, are identical. An algorithm that solves the first problem using the Star-Edge representation can be used in conjunction with a regularized intersection algorithm to implement any regularized set operation.[10] The second problem, called the "same-object" problem, was posed as an open problem by Requicha [1980] and solved for Constructive Solid Geometry (CSG) by Tilove [1981, 1984]. The complexity of Tilove's algorithm is $O(n^4)$, where $n$ is the number of intersections of surfaces used to define primitive solids in the CSG representation. The large asymptotic complexity of Tilove's algorithm is due to that fact that CSG representations can be redundant, *i.e.*, given a CSG representation $A \bigcup^* B$, there may be surfaces that define the boundaries of $A$ and $B$ that do not define the boundary of their regularized union. The redundancy tests used by Tilove's algorithm have been improved by Rossignac and Voelcker [1986], although the asymptotic complexity remains $O(n^4)$.

In order to analyze the asymptotic complexity of an algorithm that uses a B-Rep, it is necessary to describe how the B-Rep is encoded as input to the al-

[10] Kuratowski and Mostowski [1968] proved that the regular sets in conjunction with the regularized set operations form a boolean algebra. Because solids are regular sets we can express regularized union in terms of regularized intersection and complementation using one of DeMorgan's Laws:

$$R \bigcup^* S = \neg^*((\neg^* R) \bigcap^* (\neg^* S)).$$

Similarly, we can write an expression for regularized difference:

$$R -^* S = R \bigcap^* (\neg^* S).$$

gorithm. A solid's boundary is composed of connected components, and we encode each of these components as an explicit, connected B-Rep[11] containing vertex, edge, and face lists of that component. We define the size of one of these components to be the total number of vertices, faces, and edge/face intersections in that component. Using the Star-Edge representation, the size of this component is the total number of vertices, faces and *directed edges* (a directed edge describes an edge/face intersection). We also define the size of a boundary representation of a manifold solid to be the total number of vertices, faces, and edges (a manifold-solid edge is defined by the intersection of two faces; for the Star-Edge representation, each edge has exactly two directed edges).

We often cannot compare very different representations by comparing the asymptotic complexities of algorithms that use the different representations because representational sizes can very greatly (see Figure 6 on page 5). For example, the size of a Constructive Solid Geometry (CSG) representation is usually defined as the total of the number of regularized set operations and primitive solids, and hence a CSG representation of a solid is much smaller than a B-Rep of the same solid. Furthermore, algorithms for the regularized set operations are trivial using CSG, but difficult using B-Reps. However, there are many other operations which are easier to perform on B-Reps than on CSG representations. For example, we will see that the same-object problem is easier to solve using boundary representations than using Constructive Solid Geometry. Very different representations for solids can be compared by implementing equivalent algorithms using the different representations, and then by measuring computation times and other relevant factors, such as numerical error. Using this approach, Lee and Requicha [1982a] evaluated algorithms that compute volumes of solids represented in various ways.

## *3.1 Regularized Complementation for Solids*

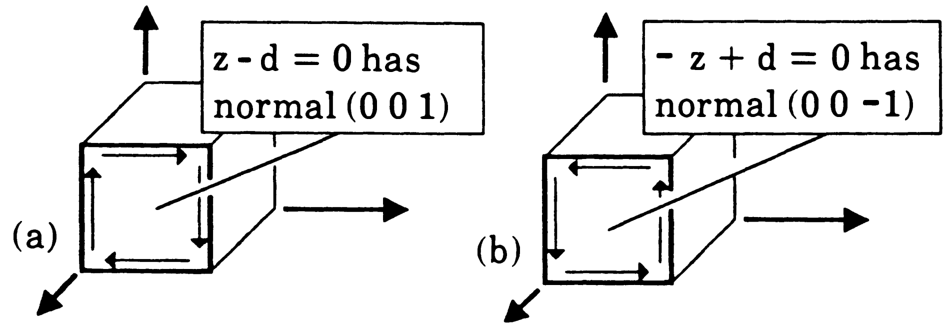
It is very easy to transform a Star-Edge representation of a solid into one for its regularized complement because a solid and its regularized complement have the same boundary.

**Lemma 3.** [Requicha, 1977] If $A$ is a regular set then $\partial(\neg^* A) = \partial A$. ■

---

[11] Recall the representational properties of the representational properties (Figure 14 on page 16) in "Chapter 2. Representing Solids".

Algorithm 3 transforms a Star-Edge representation of a solid into one for this solid's regularized complement by inverting each face's normal vector and each directed edge's orientation. (Recall that directed edges are oriented intersections of edges and their containing faces; faces have normal vectors that point to the exterior of the solid [see Figure 22].)




Solid (a) is complemented by inverting the normal vector of each face equation in its boundary represenation. This has the effect of inverting directed-edge orientations in (b).

**Figure 22. A solid and its regularized complement**

The Star-Edge representation of a solid is transformed into one for this solid's regularized complement as follows:

(1) invert the normal vector of each face;

(2) on each face, invert the radial ordering of directed edges around vertices (see Figure 19); and

(3) invert the orientation bit of each directed edge.

Consequently, the initial and terminal vertices of each directed edge are interchanged, each directed edge tangent is inverted, and the bounding directed-edge cycles of each face are reversed.

**Algorithm 3. Regularized complementation using Star-Edge encodings**

If the solid's Star-Edge representation has a length of $n$, then Algorithm 3 has a running time of $O(n)$. We can compute representations of the regularized union and difference of two solids using both Algorithm 3 and an algorithm for regularized intersection (see "Chapter 4. Intersecting Solids").

## 3.2 The Same-Object Problem for Solids

The other algorithm we examine in this chapter determines if two Star-Edge representations describe the same object. This same-object problem arises in many areas of computer-aided manufacture. For example, an algorithm that computes a machining sequence for a modeled part would need to verify that this sequence yielded an object identical to that part [Tilove, 1984].[12] The same-object problem can be solved in terms of regularized set operations,[13] but a more efficient algorithm using the Star-Edge representation is possible. This algorithm works by converting the Star-Edge representation into a canonical encoding of the minimal boundary representation (B-Rep$_{min}$) of "Chapter 2. Representing Solids" The same-object problem can then be solved efficiently because there is only one canonical B-Rep$_{min}$ encoding of a solid. We prove that the B-Rep$_{min}$ vertices, edges, and faces of a solid are unique, we show how to encode them canonically, and then we use this canonical encoding to solve the same-object problem.

**Lemma 4**. The faces, edges, and vertices of B-Rep$_{min}$ are unique.

**Proof**. Assume that $R$ and $S$ are two minimal boundary-representations of solid $X$. We prove that every face, edge, and vertex of $R$ is contained in $S$, that every face, edge, and vertex of $S$ is contained in $R$.

Without loss of generality, we argue that there must be a face, edge, or vertex of $R$ that is not found in $S$. Let us assume that there is a face $f$ of $R$ that is not contained in $S$. Then, there is a point that is in the interior of $f$ (and on the boundary of $X$). Because $f$ is not in $S$, one of the following must be true about this point: it is on some face of $S$; it is in the interior of $X$; it is in the exterior of $X$; it is at a vertex of $S$; or it is on an edge of $S$. Any of these cases imply that the neighbourhood of the point differs in $R$ and $S$, and therefore $R$ and $S$ describe different solids. Similar arguments follow for the edges and vertices of $R$ and $S$. We conclude that the faces, edges, and vertices of $R$ and $S$ are identical, and therefore, the faces, edges, and vertices of B-Rep$_{min}$ are unique. ■

---

[12] Similarity testing is a special case of congruence testing. Sugihara [1984] adapts an algorithm that tests for planar-graph isomorphism to test for the congruence of manifold solids. Atkinson [1987] tests for the congruence of point sets in $\mathbf{R}^3$, and Alt, Mehlhorn, Wagener, and Welzl [1988] discuss a variety of algorithms that test for exact and approximate congruence of point sets in $\mathbf{R}^n$. We (and Tilove [1984]) do not address this more difficult problem.

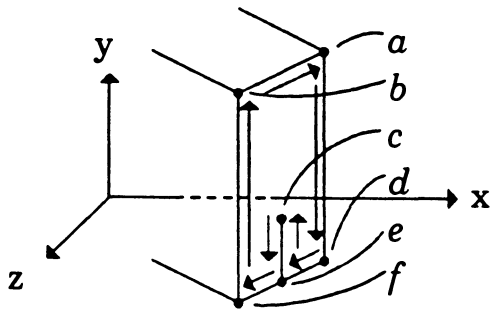
[13] Given regular sets denoted $A$ and $B$, $A = B$ iff $(A -^* B) = \phi$ and $(B -^* A) = \phi$.

Lemma 4 established that the vertices, edges, and faces of B-Rep$_{min}$ are uniquely defined. The canonical encoding of B-Rep$_{min}$ is simply a lexicographical ordering of the faces and face-boundaries of B-Rep$_{min}$. The face-ordering is directly obtained by sorting these faces using their face-equation coefficients (see Figure 23 on page 33) and the face-boundary ordering is obtained from the vertices that constitute this face boundary.

We represent each vertex $v_i$ by its coordinate triple $(x_i, y_i, z_i)$ and represent each oriented face $f_j$ by the tuple $(A_j, B_j, C_j, D_j)$, where $A_jx + B_jy + C_jz + D_j = 0$ is the oriented implicit equation of the face, and $(A_j, B_j, C_j)$ is a unit vector. We lexicographically compare two vertices or two faces using these tuples. Two tuples can be lexicographically ordered by comparing them element by element: tuple $\alpha$ is less than tuple $\beta$ if $\alpha$ and $\beta$ have identical prefixes, and the first element in the suffix of $\alpha$ is smaller than the corresponding element of $\beta$.

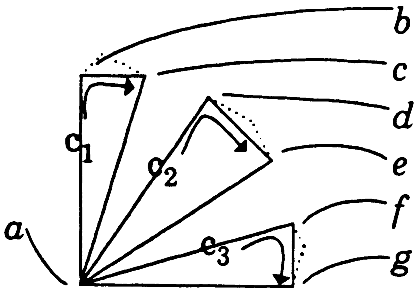
**Figure 23. Lexicographically ordering B-Rep$_{min}$ vertices and faces**

We now define a canonical encoding of the boundary of a B-Rep$_{min}$ face. Recall that the boundary of a B-Rep$_{min}$ face is defined as those edges and vertices that constitute the intersection of that face with all other faces in a B-Rep$_{min}$ representation. Because a Star-Edge face can be as general as a B-Rep$_{min}$ face (see "Chapter 2. Representing Solids") we could use the Star-Edge adjacency mechanisms, *i.e.*, bounding directed-edge cycles and isolated vertices, to describe the boundary of a B-Rep$_{min}$ face. Using directed edges complicates our same-object algorithm, and so we choose to describe a B-Rep$_{min}$ face-boundary using vertices. To convert from a directed-edge cycle to a vertex cycle, we extract the initial vertex of each directed edge; furthermore, we can treat an isolated vertex as a cycle of length zero. Each bounding vertex cycle is canonically encoded by identifying its lexicographically-smallest vertex (see Figure 23) as the first vertex of that cycle (see Figure 24 on page 34). If the lexicographically-smallest vertex is the initial vertex of more than one directed edge of the cycle, then the lexicographically-smallest of the corresponding terminal vertices is distinguished as the second vertex of the cycle. A canonical encoding of all of the vertex cycles of a B-Rep$_{min}$ face is obtained by ordering the list of vertex cycles using their lexicographically-smallest vertices. If a lexicographically-smallest vertex is shared by two or more cycles then the successor of this vertex in each cycle distinguishes and orders these cycles (see Figure 25 on page 34).




The canonical representation for the bounding cycle of this face is given by the lexicographically-smallest vertex in this cycle ($f$). Therefore, the cycle is represented as ($f,b,a,d,e,c,e$).

**Figure 24. Canonically representing a bounding cycle of a face**




If canonical representations of bounding vertex-cycles $c_1$, $c_2$, and $c_3$, are ($a,b,c$), ($a,d,e$), and ($a,f,g$), respectively, these three cycles are ordered by lexicographically ordering coordinates of $b$, $d$, and $f$.

**Figure 25. Canonically representing the boundary of a face**

Using Lemma 4 and the canonical encoding described for the boundary of a B-Rep$_{min}$ face, we define the canonical encoding of B-Rep$_{min}$. Each component of the surface of a solid is canonically encoded by a list of lexicographically-ordered B-Rep$_{min}$ faces. Each face is represented by its (normalized) oriented plane equation and by the lexicographically-ordered list of lexicographically-ordered bounding vertex cycles of the face boundary. Finally, we canonically encode the list of surface-component encodings by lexicographically ordering them using the smallest vertex on each component.

In order to solve the same-object problem for two boundary representations, convert to B-Rep$_{min}$, encode canonically, and then compare the two canonical encodings of B-Rep$_{min}$. As this comparison can be done in linear time, the time required to solve the same-object problem using the input boundary representations is bounded by the time required to convert the two input representations to B-Rep$_{min}$ and canonically encode them. We now examine that conversion and encoding process for the Star-Edge representation.

If an instance of the Star-Edge representation has faces that are the same as the connected components of a B-Rep$_{min}$ face, then conversion of that instance of the Star-Edge representation is relatively easy. In general, Star-Edge faces can be much simpler than the connected components of B-Rep$_{min}$, and then converting the Star-Edge representation to B-Rep$_{min}$ is more tedious. Thus we need an algorithm that finds a canonical encoding of the B-Rep$_{min}$ representation of a solid, given the Star-Edge representation of that solid.

Constructing the canonical B-Rep$_{min}$ encoding from a Star-Edge representation of a solid is done first by combining coplanar faces and then by deleting all edges and vertices that are not boundary elements of B-Rep$_{min}$. This construction is described by Algorithm 4, which has an asymptotic running time of $O(V + D \log D + F \log F)$, where $V$, $D$, and $F$ are the numbers of vertices, directed edges, and faces, respectively, in the Star-Edge representation.

The following algorithm constructs the canonical B-Rep$_{min}$ encoding from the Star-Edge representation of a solid:

(1) Lexicographically order the faces of the Star-Edge representation. Mark all faces with the same oriented face-equation as belonging to one B-Rep$_{min}$ face.

(2) If all faces adjacent to an edge belong to the same B-Rep$_{min}$ face, delete the edge and the associated directed edges, and join the affected bounding directed-edge cycles.

(3) If every edge adjacent to a vertex belongs to every B-Rep$_{min}$ face incident to the vertex, delete the vertex, join the edges (hence directed edges) adjacent to the vertex, and join the affected bounding directed-edge cycles.

(4) Construct the lexicographically-ordered vertex cycles from both the bounding-directed edge cycles and the isolated vertices of each face boundary.

If there are $D$ directed edges, $F$ faces, and $V$ vertices, respectively, in the Star-Edge representation, then Step (1) can be carried out in time $O(F \log F)$, Steps (2) and (3) require time $O(D + V)$, and Step (4) requires time $O(D \log D)$. Thus B-Rep$_{min}$ can be extracted from a Star-Edge representation in a total time of $O(V + D \log D + F \log F)$.

**Algorithm 4. Converting from Star-Edge to canonical B-Rep$_{min}$ Encoding**

We now prove that the problem of computing B-Rep$_{min}$ from an explicit, connected boundary representation with $F$ faces has an $\Omega(F \log F)$ lower bound. The proof of Theorem 5 shows that the problem of determining if the elements in a list of size $n$ are unique, can be solved by constructing the canonical B-Rep$_{min}$ encoding from an explicit, connected B-Rep of a particular solid. The model of computation used, the algebraic-decision-tree model, is sufficiently powerful for the algorithms of this thesis.

**Theorem 5.** The problem of constructing the canonical B-Rep$_{min}$ encoding from an explicit, connected representation (of a solid) with $F$ faces has an $\Omega(F \log F)$ lower bound in the algebraic-decision-tree model.

**Proof.** Given a list $a$ of $n$ numbers $(a_1, a_2, \ldots, a_n)$, the problem of determining if the elements of $a$ are unique is transformed into the problem of constructing the canonical B-Rep$_{min}$ encoding from an explicit, connected B-Rep with $2n + 4$ faces, $6n + 6$ edges, and $4n + 4$ vertices, which describes a "staircase." This transformation is done by constructing this staircase along the x-axis between $x = 0$ and $x = n$, with the height of a "step" corresponding to an element in $a$. (Without loss of generality, we assume that no two *consecutive* elements in $a$ are identical and that the first and last elements are zero with all the others positive.) Finally, the elements of $a$ are unique if and only if the staircase has exactly $2n + 4$ faces. We now describe how to construct the staircase.

An explicit, connected representation of this staircase has four types of faces: the supporting floor, the top of a step, the front of a step, and the two sides. We give the vertices of each of these faces as a clockwise cycle. (We will not represent these faces in the canonical form of Figure 24 on page 34 and Figure 25 on page 34: although we could easily do this without penalty.) The supporting floor, or *bottom* of this staircase (see Figure 26), is a rectangle on the plane ($y = 0$), with a bounding vertex cycle:

$$((0,0,0), (0,0,1), (n,0,1), (n,0,0)).$$

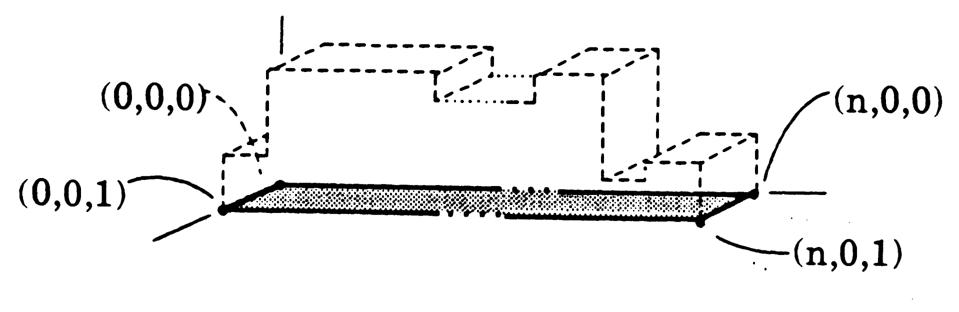



**Figure 26. Bottom of the staircase**

The top of a step, or *tread* (see Figure 27 on page 37), is a square in the plane ($y = a_i$), *i.e.*, the $i^{th}$ tread of this staircase has height given by the $i^{th}$ element of $a$. The $i^{th}$ tread has a bounding vertex cycle:

$$((i-1, a_i, 0), (i, a_i, 0), (i, a_i, 1), (i-1, a_i, 1)).$$

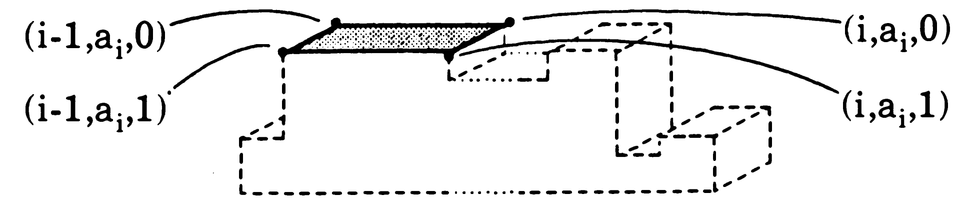



**Figure 27. Tread of the staircase**

The front of a step, or *riser* (see Figure 28), is a rectangle in the plane $(x = i)$, with a bounding vertex cycle:

$$((i, a_i, 1), (i, a_i, 0), (i, a_{i+1}, 0), (i, a_{i+1}, 1)).$$

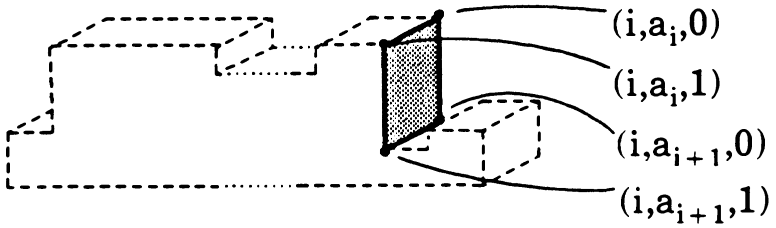



**Figure 28. Riser of the staircase**

Finally, the two *sides* of this staircase, shown in Figure 29 on page 38, are more complicated. The side on the plane $(z = 0)$ has a bounding vertex cycle:

$$((n,0,0), (n, a_n,0), (n-1, a_n,0), \ldots, (1, a_1, 0), (0, a_1,0), (0,0,0)); \text{ and}$$

the side on the plane $(z = 1)$ has a bounding vertex cycle:

$$((0,0,1), (0, a_1,1), (1, a_1,1), \ldots, (n-1, a_n,1), (n, a_n,1), (n,0,1)).$$

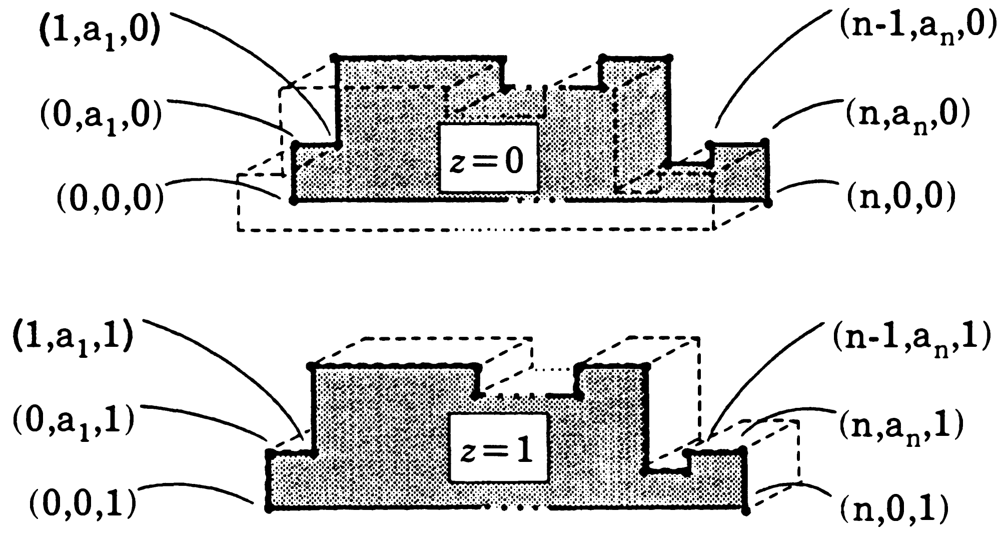



**Figure 29. Sides of the staircase**

The completed staircase is the manifold solid shown in Figure 30.

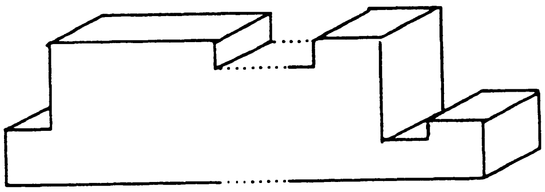


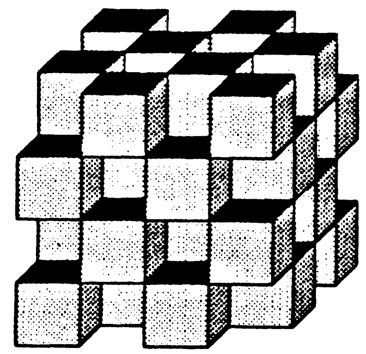
**Figure 30. The completed staircase:** A boundary representation of the staircase contains $4n + 4$ vertices ($2n + 2$ vertices along each side of the staircase), $2n + 4$ faces ($n$ treads, $n + 1$ risers, two sides, and one bottom), and $6n + 6$ edges (4 bottom edges, 2 vertical edges in each riser, and 4 edges on each tread).

In order to complete the proof of this theorem, we must show how an algorithm that constructs the canonical B-Rep$_{min}$ encoding from an explicit, connected representation is used to determine if the elements of the list $a$ are unique. First we observe that an explicit, connected representation of the staircase is linear in the size of $a$ because it has (from Figure 30) $12n + 14$ boundary elements.[14] More importantly, there are $2n + 4$ faces in this representation. Next we observe that there are duplicates in $a$ if and only if there are coplanar treads in the staircase, and that an algorithm, by constructing the canonical B-Rep$_{min}$, encoding, combines these coplanar treads.

[14] Note that we may have added a linear number of elements to $a$ in order to ensure that no two adjacent elements of $a$ were identical. This technical condition makes the staircase construction simpler.

Thus, the elements of $a$ are unique if and only if there are $2n + 4$ faces in the B-Rep$_{\min}$ representation of the staircase. Because the problem of deleting duplicates from a list of size $n$ has a lower bound of $\Omega(n \log n)$ in the algebraic-decision-tree model [Preparata and Shamos, 1985, pg. 186], the construction of B-Rep$_{\min}$ from an explicit, connected representation of the staircase requires time $O(n \log n)$. Therefore, the problem of constructing the canonical B-Rep$_{\min}$ encoding from an explicit, connected representation of a solid with $F$ faces has a lower bound of $\Omega(F \log F)$. ■

Theorem 5 gives a lower bound of $\Omega(F \log F)$ for the problem of constructing the canonical B-Rep$_{\min}$ encoding from a connected, explicit representation of a solid with $F$ faces; and Algorithm 4 on page 35 solves this problem in time $O(V + D \log D + F \log F)$, where $F$, $D$, and $V$ are the number of faces, directed edges, and vertices, respectively, in the Star-Edge representation. For manifold solids with polygonal faces, $D < 6F$ and $V < 4F$, and so Algorithm 4 on page 35 is optimal. However, it is possible to construct a Star-Edge representation of a solid that has few faces, but a great many vertices and directed edges. For example, Figure 31 describes a solid with $O(n)$ Star-Edge faces, $O(n^3)$ directed edges, and $O(n^3)$ vertices.



For $n = 4$, the solid is made from 32 cubes; the Star-Edge representation for the solid can have:
24 faces, 115 vertices, and 768 directed edges.

**Figure 31. A solid with many more edges and vertices than faces:** The solid is made by joining $\frac{n^3}{2}$ cubes along edges. The Star-Edge representation of the solid has $6n$ faces in the solid ($n$ faces in each of the six principal direction), $12n^3$ directed edges (each cube has 24 directed edges), and $(n + 1)^3 - 2n - 2$ vertices (in a principal direction, each of the $n + 1$ planes has $n^2 + 2n - 1$ vertices).

The same-object algorithm, *i.e.*, comparison of two canonical B-Rep$_{\min}$ encodings constructed using Algorithm 4 on page 35, can be implemented using exact arithmetic (*e.g.*, rational arithmetic) in the straightforward way. However, if we wish to implement the same-object algorithm using finite-precision (*e.g.*, floating-point arithmetic), we must be more careful. Because finite-precision arithmetic implements equality using proximity (within a tolerance $\varepsilon$), three problems arise:

(1) Two vertices with proximal $x$-coordinates cannot be sorted using a finite-precision lexicographic comparison; consequently, it may be impossible to identify the lexicographically-smallest vertex of a bounding vertex cycle of a face. For the same reason, it may not be possible to lexicographically-order two different vertex cycles on the same face.

(2) Two faces whose normalized face-equations have proximal $x$-coefficients cannot be lexicographically-ordered using finite-precision arithmetic.

(3) If two faces lie on oriented planes that are almost identical, it may be impossible to determine if the two faces should be coalesced into a single B-Rep$_{min}$ face.

These three problems illustrate that it may not be possible to construct the canonical B-Rep$_{min}$ encoding of a solid using finite-precision arithmetic. Algorithm 4 on page 35 can always make arbitrary choices to solve these three problems, but then the comparison phase of the same-object algorithm will simply report that the two solids are different. By modifying the B-Rep$_{min}$ encoding to explicitly describe ambiguities that result from implementing the ordering calculations with finite-precision, error due to the three problems can be reduced.

Problem (1) is solved by identifying, for each cycle, the list of "possible" lexicographically-smallest vertices with proximal $x$-coordinates; and the B-Rep$_{min}$ encoding of such a cycle is augmented with the list. The comparison phase of the same-object algorithm compares two such augmented cycles by comparing all possible cycle-pairs. Similarly, if the cycles of a B-Rep$_{min}$ face cannot be lexicographically-ordered, the comparison phase of the same-object algorithm compares B-Rep$_{min}$ faces by comparing all possible pairs of cycles.

Problem (2) is solved similarly to Problem (1); the B-Rep$_{min}$ encoding of a solid is modified to contain clusters of faces with proximal $x$-coefficients. The comparison phase of the same-object algorithm compares two such clusters of faces by comparing all possible face-pairs.

To alleviate Problem (3), we make a minimum-feature-size assumption [Segal and Séquin, 1985] about the input boundary representations: Two faces that are coplanar to a tolerance $\varepsilon$ and that share a vertex $v$ are defined to be coplanar. The idea behind this assumption is the following: two of the three planes that define $v$ are nearly coplanar, and so the coordinates of $v$ contain a large error. When two faces are coalesced to form a single B-Rep$_{min}$ face, their face-equations are combined to form an equation with interval-coefficients. For example, two vectors of coefficients, $(a, b, c, d)$ and $(a', b', c', d')$, are combined to form the vector of interval-coefficients

$$([a \min a', a \max a'],\ [b \min b', b \max b'],\ [c \min c', c \max c'],\ [d \min d', d \max d']);$$

the comparison phase of the same-object algorithm compares two lexicographically-ordered lists of face-equations by comparing interval-coefficients.

Our discussion of the problems in a finite-precision implementation of the same-object algorithm provide some context for the remainder of this thesis. If we choose to implement a geometric algorithm to use exact computation, then the implementation will be correct, although inefficient. Conversely, if we implement the algorithm using finite-precision arithmetic, then the implementation will be efficient, although probably incorrect. In the next two chapters, we discuss both a solid-intersection algorithm and its finite-precision implementation. This implementation uses symbolic inference techniques to alleviate errors due to finite precision.

# Chapter 4. Intersecting Solids

In this chapter, a new regularized intersection algorithm for solids is described, analyzed, and proved correct. Given Star-Edge representations of solids $A$ and $B$, the algorithm computes a Star-Edge representation of $A \cap^* B$. The new intersection algorithm has been implemented as described here, using finite-precision incidence tests to be described in "Chapter 5. Incidence Tests" (see also Appendices B-E).

Few attempts have been made to develop correct, efficient algorithms that intersect solids, although such algorithms have been presented for convex polyhedra. Muller and Preparata [1978] describe an algorithm that intersects two convex polyhedra, and Brown [1978] and Preparata and Muller [1979] use this algorithm to intersect a collection of half-spaces. Hertel, Mantyla, Mehlhorn, and Nievergelt [1984] describe algorithms that compute the union, intersection, or difference of convex polyhedra. All of these algorithms use a representation of Preparata and Muller [1978], called the *doubly-connected edge list* (*DCEL*), which is almost identical to the Winged-Edge representation of Baumgart [1972] (described in "Chapter 3. Some Representational Results").

Muller and Preparata [1978] describe an algorithm that, given DCEL representations of convex polyhedra $P$ and $Q$, constructs a DCEL representation of $P \cap Q$ in time $O(V \log V)$, where $V$ is the total number of vertices in $P$ and $Q$. If both $P$ and $Q$ contain the origin, their intersection can be written as the intersection of half-spaces of the form $a_i x + b_i y + c_i z \leq 1$. The *dual* of each half-space is defined to be the point $(a_i, b_i, c_i)$, and the dual of a convex polyhedron is defined to be the convex hull of the duals of its defining half-spaces. Muller and Preparata [1978] prove that if $P$ and $Q$ are convex polyhedra that contain the origin, then $P \cap Q$ is the dual of the convex hull of the union of the duals of $P$ and $Q$. Thus $P \cap Q$ is found by: (i) finding a point $a$ in the interior of $P \cap Q$; (ii) translating $P$ and $Q$ so that $a$ is at the origin; (iii) computing the points that are the duals of the (translated) defining half-spaces of $P$ and $Q$; (iv) finding the convex hull of the points found in Step (iv); and (v) obtaining $P \cap Q$ by applying the inverse translation of Step (ii) to the dual of the convex hull found in Step (iv). Muller and Preparata [1978] give a detailed algorithm for Step (i) that uses time $O(V \log V)$ to either find $a$ or determine that $P \bigcap Q = \phi$. They also use the convex-hull algorithm of Preparata and Hong [1977] in Step (iv), which finds the convex hull of $n$ points in time $O(n \log n)$.

The intersection algorithm of Muller and Preparata [1978] is used by Brown [1978] and Preparata and Muller [1979]. to intersect a set of $F$ half-spaces in time $O(F \log F)$ First, the $F$ half-spaces are partitioned into two sets: set $H^+$ contains half-spaces of the form $a_i x + b_i y + c_i z \leq 1$, and set $H^-$ contains half-spaces of the form $a_j x + b_j y + c_j z \leq -1$. Then the duals of the half-spaces of $H^+$ and $H^-$ are found: the dual of a half-space of $H^-$ is point $(a_i, b_i, c_i)$, and the dual of a half-space of $H^-$ is point $(-a_j, -b_j, -c_j)$. The $F$ half-spaces intersect if and only if there is a plane that separates the duals of the half-spaces of $H^-$ from the duals of the half-spaces of $H^-$. (Preparata and Shamos [1985, pg. 291] give an algorithm that uses time $O[F]$. to either find a separating plane or determine that one does not exist.) Using homogeneous coordinates, the rotation that maps the separating plane to the plane at infinity also transforms every half-space of $H^-$ and $H^-$ to contain the origin. Finally, the $F$ rotated half-spaces are intersected using the method of Muller and Preparata [1978], and the inverse rotation is applied to the resulting convex polyhedron. In total, the algorithms of Brown [1978] and Preparata and Muller [1979] require time $O(F \log F)$ to produce a DCEL representation of the intersection of $F$ half-spaces.

Hertel, Mantyla, Mehlhorn, and Nievergelt [1984] describe algorithms that compute the intersection, union, or difference of convex polyhedra $P_0$ and $P_1$ in time $O(V \log V)$, where $V$ is the total number of vertices in $P_0$ and $P_1$. (They represent the polyhedra using the DCEL representation). A plane, orthogonal to an arbitrarily chosen line, is swept along that line. As the plane passes through $P_0$ and $P_1$, balanced binary trees $C_0$ and $C_1$ keep track of the edges and faces of $P_0$ and $P_1$, respectively, intersected by the plane. The direction of sweeping induces an order, called a *sweep-order*, on the vertices of $P_0$ and $P_1$. An edge $(v,w)$ of $P_i$ is defined to *start* at vertex $v$ if $v$ precedes $w$ in sweep-order, and a face of $P_i$ is defined to start at $v$ if $v$ precedes every other vertex of the face in sweep-order. Faces and edges of $P_i$ that *end* at $v$ are similarly defined. At each vertex $v$ of $P_i$, in sweep-order, the following procedure is used to find some of the intersections of edges of $P_i$ with faces of $P_{1-i}$:

(1) First $C_i$ is updated. Added to $C_i$ are the edges and faces of $P_i$ that start at $v$, and deleted from $C_i$ are the edges and faces of $P_i$ that end at $v$.

(2) Then the edges of $C_i$ that start at $v$ are intersected with the faces in $C_{1-i}$, and the intersection points, if any, are recorded.

(3) Finally, for each face $f$ of $C_i$ that starts at $v$, if no intersections of edges of $f$ with the faces of $C_{1-i}$ were found by Step (2), the interior of $f$ could be pierced by the faces of $C_{1-i}$. To determine whether this is so, an arbitrary edge of $C_{1-i}$ is intersected with $f$, and the intersection point, if any, is recorded.

The authors show that the above procedure requires total time $O(V \log V)$ to find a sufficient number of intersection points so that a boundary representation of $P_0 \bigcap P_1$, $P_0 \bigcup P_1$, or $P_0 - P_1$, can be subsequently constructed in time $O(V)$. The authors conclude that although the details of their intersection algorithm are complicated, the algorithm is conceptually simple, and they also suggest that the space-sweep paradigm does not have the same power in $\mathbf{R}^3$ as plane-sweeps have in $\mathbf{R}^2$.

The algorithms described above are not so relevant to the problem of solid regularized-intersection because they exploit the properties of convex polyhedra. The problems of developing correct and efficient algorithms for regularized set operations of solids are summarized by Mantyla [1986]:

> "... a set operations algorithm must be able to treat all possible kinds of geometric intersections that may appear between faces, edges, and vertices of the two objects. The proper treatment of all cases easily leads to a very hairy case analysis. Second, the very case analysis must be based on various tests for overlap, coplanarity, and intersection, which are difficult to perform robustly in the presence of numerical errors."

Mantyla describes the details (corresponding to the "hairy case analysis" above) of incidence tests used in union, intersection, and difference algorithms for *manifold* solids represented using the Winged-Edge representation. Laidlaw, Trumbore, and Hughes [1986] present a simple, albeit asymptotically slow algorithm for the regularized intersection of two solids with connected boundary. Laidlaw, Trumbore, and Hughes claim that a proof of correctness of their algorithm is possible, although they do not offer one. Paoluzzi, Ramella, and Santarelli [1986] present an algorithm that computes the regularized union of two solids with connected boundary, but their discussions of special cases and asymptotic complexity are sketchy (these two algorithms are discussed in "Asymptotic Complexity of the Intersection Algorithm" on page 50). Segal and Séquin [1988] present an algorithm that intersects manifold solids, but they do not present an analysis of asymptotic complexity nor do they discuss special cases in any detail.

The presentation of the new intersection algorithm of this chapter improves upon the work cited above by giving an asymptotic analysis of complexity which encompasses all of the special cases. (Of the algorithms cited above, the only one that discusses special cases in detail is that of Mantyla's.) The new intersection algorithm improves on Mantyla's algorithm by generalizing the domain to solids.

## *4.1 Overview of the Intersection Algorithm*

Star-Edge representations of $A$ and $B$ are used by the new intersection algorithm to construct a Star-Edge representation of $A \cap^* B$. The boundary of a solid is not necessarily connected, and so each *shell*, or connected component of the solid boundary, will be represented by lists of the vertices, edges, and faces of the Star-Edge representation of that shell.[15] Figure 32 is a paradigm for intersecting objects that have several boundary components and are represented by their boundaries.

Given regular but not necessarily connected objects $A$ and $B$ represented by their boundaries, a boundary representation of $A \cap^* B$ is produced as follows:

(1) Intersect the shells of $A$ with the shells of $B$, and assemble pieces of the intersecting shells to obtain some of the shells of $A \cap^* B$.

(2) Obtain the remaining shells of $A \cap^* B$ by taking those shells of $A$ (resp. $B$) contained entirely in the interior of $B$ (resp. $A$).

Step (1) is complicated by extra processing to ensure that the result of the intersection is a regular set.

**Figure 32. Standard computation of $A \cap^* B$**

Before showing how Figure 32 might lead to an algorithm that intersects solids $A$ and $B$, we examine a simpler case. In two dimensions, $A$ and $B$ might be polygons with polygonal holes, and Figure 32 would say: (1) intersect the boundary polygons of $A$ with the boundary polygons of $B$, yielding a collection of intersection points that are used to trace out some of the boundary polygons of $A \cap^* B$; and (2) obtain the remaining boundary polygons of $A \cap^* B$ by taking

[15] We will use some shorthand for brevity. For example, when we refer to a "face of a solid" we mean to refer to a "face of the Star-Edge representation of a shell of a solid."

every boundary polygon of $A$ that is in the interior of $B$ and every boundary polygon of $B$ that is in the interior of $A$. Regularization of the intersection might be necessary if a vertex of a boundary polygon of $A$ (resp. $B$) is incident with an edge of a boundary polygon of $B$ (resp. $A$). A boundary representation of $A \cap^* B$ contains four different types of edges. These are edges of $A \cap^* B$ contained in edges of $A$ (resp. $B$) that intersect polygons of $B$ (resp. $A$), and edges of $A$ (resp. $B$) contained within the interior of $B$ (resp. $A$). (We could also distinguish edges of $A \cap^* B$ contained both in edges of $A$ and edges of $B$.)

Algorithms that use the paradigm of Figure 32 on page 45 to intersect solids $A$ and $B$ first intersect faces of $A$ with faces of $B$ to obtain line segments and points that are used to trace the boundaries of the faces of some of the shells of $A \cap^* B$; then the remaining shells of $A \cap^* B$ are obtained by taking the shells of $A$ ($B$) that are contained in the interior of $B$ ($A$). Just as in the two-dimensional example above, a boundary representation of $A \cap^* B$ contains four different types of faces. These four types correspond to faces of $A \cap^* B$ that are:

(i) contained in faces of $A$ that intersect the surface of $B$;

(ii) contained in faces of $B$ that intersect the surface of $A$;

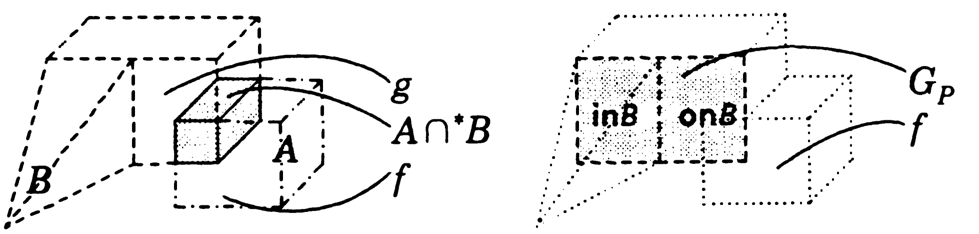
(iii) faces of $A$ that are contained in the interior of $B$; and

(iv) faces of $B$ that are contained in the interior of $A$.

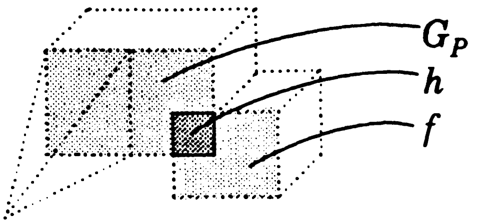
We distinguish faces of $A \cap^* B$ that are contained in both faces of $A$ and $B$ to be faces of Type (i).

We now show how the four types of faces of $A \cap^* B$ as defined above are constructed by the new intersection algorithm of this thesis. The algorithm has six steps, which are performed in turn. Step (1) of the new algorithm intersects $B$ with the oriented plane $P$ of each face $f$ of $A$, yielding a partition ($G_P$) of $P$ into two-dimensional sets that are in the interior of $B$, in faces of $B$ oriented identically to $P$, or in the exterior of $B$ as shown in Figure 33(a). Faces of $B$ oriented oppositely to $P$ are considered to be in the exterior of $B$. Concurrently, $G_P$ is augmented with edges and vertices of $B$ that lie on $P$. Step (2) of the algorithm finds all faces of Type (i) and some faces of Types (iii). Every face $f$ of $A$ is intersected with the corresponding $G_P$. If $f$ contains points that are on the boundary of $B$, the Type (i) faces contained in $f$ are found (see Figure 33[b]); otherwise, if $f$ is contained in the interior of $B$, then $f$ is a Type (iii) face itself.

Step (3) of the algorithm finds additional Type (iii) faces.[16] If a face $f$ from Step (2) contains Type (i) faces or is a Type (iii) face itself, then $f$ might have neighbouring faces that are in the interior of $B$. These neighbouring faces are Type (iii) faces, which can in turn neighbour additional Type (iii) faces. Beginning with $f$, a breadth-first search examines the edges and vertices of faces of $A$ to find Type (iii) faces that were not found by Step (1) and have not yet been found by Step (2) (not shown in Figure 33).




(a) The intersection of solid ($B$) with the cube ($A$) is a cuboid. Plane $P$ of the front face $f$ of $A$ is coincident with face $g$ of $B$, and the cross-section $G_P$ defined by $B \cap P$ contains $g$ and a two-dimensional set in the interior of $B$.




(b) The intersection of $f$ with the points of $G_P$ that are **in**$B$ or **on**$B$ is the square area $h$ that is a face of $A \cap^* B$.

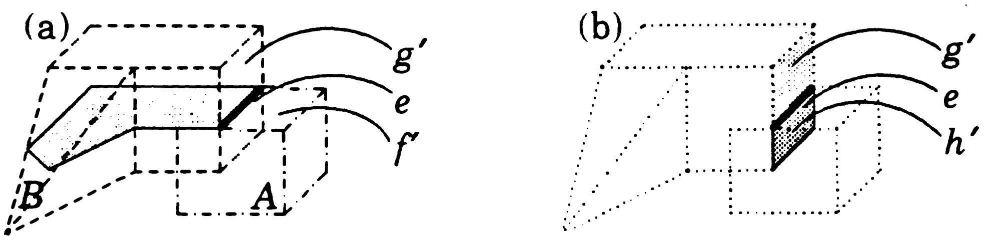
**Figure 33. Steps (1) and (2) of the new intersection algorithm**

Step (4) of the intersection algorithm finds all Type (ii) faces, and Step (5) finds some Type (iv) faces. The Type (ii) and Type (iv) faces could be constructed by interchanging the roles of solids $A$ and $B$ and repeating the first three steps of the algorithm, but there is a more direct way to generate these faces. By definition, every face $g$ of $B$ that contains Type (ii) faces must intersect the boundary of $A$, and so Step (2) of the intersection algorithm must have found the points and line segments of the boundary of $A$ that are contained in $g$. We can

[16] Step (3) is necessary if solid $B$ is unbounded, *e.g.*, if $B$ is the complement of a cube.

use these points and line-segments, which are edges and vertices of $A \cap^* B$, to find the Type (ii) faces contained in $g$ (Figure 34 on page 48). Note that two-dimensional intersections of $g$ with the boundary of $A$ are contained in Type (i) faces, and so we only consider Type (ii) faces that intersect the interior of $A$. Step (5) of the intersection algorithm obtains some Type (iv) faces by searching for faces of $B$ that lie entirely in the interior of $A$, beginning with the neighbouring faces of $g$. All faces thus obtained are Type (iv) faces (not shown in Figure 34).

Finally, Step (6) of the intersection algorithm obtains the remaining Type (iii) and Type (iv) faces. The remaining Type (iii) faces belong to shells of $A$ that did not contribute any boundary points to the faces of $A \cap^* B$ found in Step (2). The remaining Type (iv) faces belong to shells of $B$ that did not contribute any boundary points to the faces of $A \cap^* B$ found in Step (4) (not shown in Figure 34).




In (a), the plane of the top face $f'$ of $A$ intersects the rightmost face $g'$ of $B$ in line segment $e$. Because $e$ is an edge of $A \cap^* B$, $e$ is used in (b) to construct the face $h'$ of $A \cap^* B$ contained in $g'$

**Figure 34. Step (4) of the new intersection algorithm**

The new intersection algorithm is given as Algorithm 5 on page 49. The remainder of this chapter fills in the details of the algorithm. For example, we will see how the intersection is regularized by discarding certain intersections of the surfaces of $A$ and $B$.

A Star-Edge representation of $A \cap^* B$ is constructed from Star-Edge representations of $A$ and $B$ as follows:

(1) For each face $f$ of $A$, intersect $B$ with the oriented plane $P$ of $f$, yielding $G_P$, a partition of $P$ into two-dimensional sets that are in the interior of $B$ (**in**$B$), in faces of $B$ oriented identically to $P$ (**on**$B$), and in the exterior of $B$. Faces of $B$ oriented oppositely to $P$ are considered to be in the exterior of $B$. Also, $G_P$ is augmented with edges and vertices of $B$ that lie on $P$ (see Figure 33[a]).

(2) For each face $f$ of $A$, intersect $f$ with subsets of $G_P$ that are **in**$B$ and **on**$B$, yielding vertices, edges, and faces of $A \cap^* B$ contained in $f$ (see Figure 33[b]).

(3) Find the faces of $A$ that are in the interior of $B$ and belong to shells of $A$ that contain the faces of $A \cap^* B$ found in Step (2). The faces of $A$ obtained by this search are also faces of $A \cap^* B$ (not shown in Figure 33).

(4) For each face $g$ of $B$, find the edges and vertices of $A \cap^* B$ constructed by Step (2) that are also contained in $g$, and use these edges and vertices to construct faces of $A \cap^* B$ contained in $g$ that intersect the interior of $A$ (see Figure 34).

(5) Find the faces of $B$ that are in the interior of $A$ and belong to shells of $B$ that contain the faces of $A \cap^* B$ found in Step (4). The faces of $B$ obtained by this search are also faces of $A \cap^* B$ (not shown in Figure 34).

(6) Finally, the remaining shells of $A \cap^* B$ are obtained from the set $A_0$ of shells of $A$ that do not contribute any boundary points to the faces of $A \cap^* B$ found in Step (2), and the set $B_0$ of shells of $B$ that do not contribute any boundary points to the faces of $A \cap^* B$ found in Step (4). The shells of $A_0$ ($B_0$) that are contained in the interior of $B$ ($A$) are taken to be shells of $A \cap^* B$.

In each of Steps (1)-(2) and (4), extra processing is done to regularize the intersection of $A$ and $B$.

**Algorithm 5. The new regularized intersection algorithm**

In the description of the algorithm to follow, we assume that computations can be performed exactly and that numerical quantities can be tested for equality.[17] Then in "Chapter 5. Incidence Tests" a finite-precision implementation

[17] It is possible to implement algorithms like the new regularized intersection algorithm to use rational arithmetic [O'Connor and Rossignac, 1987], although this has not been done. Given an algorithm that sorts vectors radially around a point on a plane (see Appendix C), the intersection algorithm could be implemented to use rational arithmetic.

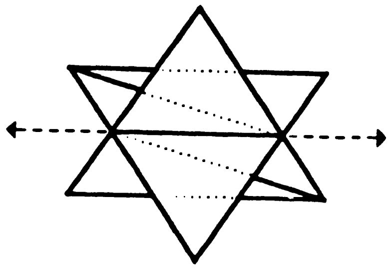
of the incidence tests used by the algorithm is discussed. Before describing the new intersection algorithm, the main result of this chapter is stated.

## 4.2 Asymptotic Complexity of the Intersection Algorithm

**Theorem 6.** Given Star-Edge representations of the shells of $A$ and $B$ with $D_A$ and $D_B$ directed edges respectively, a Star-Edge representation of $A \cap^* B$, which has total size $O(D_A D_B)$, can be constructed in time $O(D_A D_B \log[D_A D_B])$. ■

The numbers of faces, edges, directed edges, and vertices in a Star-Edge representation of an object $S$ are denoted by $F_S$, $E_S$, $D_S$, and $V_S$, respectively; and the size $|S|$ of $S$ is defined as $F_S + D_S + V_S$. For example, if $S = f$ is a face then $D_f$ is the number of directed edges of $f$, and $|f| = 1 + D_f + V_f$. The simple expressions for the asymptotic complexities in Theorem 6 are obtained by simplifying using the relationships among $F_A$, $E_A$, $D_A$, , and $V_A$ proved in "Some Combinatorial Properties of the Star-Edge Representation" on page 22. The remainder of this chapter proves Theorem 6 and establishes the correctness of the intersection algorithm.

Other researchers claim that their algorithms implement regularized set operations in time that is quadratic in the size of their input boundary representations, but when carefully analyzed, their algorithms are significantly slower. The algorithms of Laidlaw, Trumbore, and Hughes [1986] and Paoluzzi, Ramella, and Santarelli [1986] intersect solids $A$ and $B$ by intersecting each face of $A$ with each face of $B$. Pairs of intersecting faces are subdivided into smaller faces that intersect only at edges and vertices. The representation of Laidlaw, Trumbore, and Hughes requires faces with convex polygonal boundaries and so the subdivision produces convex polygonal faces. Similarly, Paoluzzi, Ramella, and Santarelli subdivide faces into triangular faces (see Figure 35 on page 51). In either case the subdivision is described by a simple case-analysis and so one would expect the subdivision to be efficient for these two representations. However, each subdivision increases the number of faces of $A$ and $B$ so that the subdivision requires time $O(D_A^2 D_B^2)$. Thus, the algorithm of Laidlaw, Trumbore, and Hughes and the algorithm of Paoluzzi, Ramella, and Santarelli require time $O(D_A^2 D_B^2)$ to compute boundary representations of $A \cap^* B$ and $A \cup^* B$, respectively, from boundary representations of $A$ and $B$. Thus these two algorithms are in fact asymptotically slower than the new intersection algorithm of this thesis.



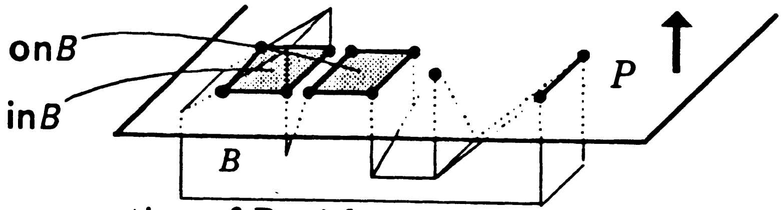
Two intersecting triangles are subdivided by the intersection line and by additional segments that partition the subdivisions into triangles.

**Figure 35. Triangulating the subdivision of intersecting faces**

## *4.3 Computing Cross-Sections of Solid B*

Given solids $A$ and $B$, Step (1) of Algorithm 5 on page 49 intersects $B$ with the oriented plane $P$ of each face $f$ of $A$. The result of each intersection, called $G_P$, is a partition of $P$ into two-dimensional sets that are inside, outside, or on the surface of $B$, denoted **in**$B$, **out**$B$, and **on**$B$, respectively (see Figure 36 on page 52). Because we are computing a regularized intersection of $A$ with $B$, the interiors of faces of $B$ coincident with $P$ but whose planes are oriented oppositely to $P$ are **out**$B$, and faces of $B$ whose planes are identical to $P$ are **on**$B$. We also augment $G_P$ with edges and vertices of $B$ that lie on $P$.

Just as directed edges and vertices represent the boundary of Star-Edge faces, directed edges and vertices are used to represent the boundaries of the connected two-dimensional **in**$B$ and **on**$B$ sets of $G_P$ (recall the definitions of Star-Edge and B-Rep$_{min}$ faces from "Chapter 2. Representing Solids"). As described in Algorithm 6 on page 52, certain edges and vertices of $B$ that coincide with $P$ are edges and vertices of $G_P$; and other edges and vertices of $G_P$ result from transverse intersections of $P$ with faces and edges, respectively, of $B$ (see also Figure 36 on page 52). We orient the edges of $G_P$ so that on $P$, the area to the right of the directed edge is either **in**$B$ or **on**$B$ when traversing the directed edge from its initial to its terminal vertex. If the interior of $B$ is not bounded, *e.g.*, if $B$ is the complement of a cube, then one of the **in**$B$ point sets of $G_P$ is not bounded, but we can still represent the (bounded) boundary of this unbounded point set using directed edges and vertices.




Cross section of $B$ with oriented plane $P$ partitions $P$ into line segments and points that are on the boundary of $B$, and two-dimensional sets that are in the interior of $B$, on the boundary of $B$, and in the exterior of $B$.

**Figure 36. The cross-section $G_P$**

For each face $g$ of $B$, find the two-dimensional sets, line-segments, and points that are the intersection of $g$ with oriented plane $P$ as follows:

(1) Find the isolated vertices of $g$ that lie in $P$. For each edge $e$ of $g$, compute the intersection of $e$ with $P$. There are four cases: $e$ does not intersect $P$; the interior of $e$ intersects $P$ at a point; $e$ lies in $P$ (the intersection points are taken to be the endpoints of $e$); or an endpoint of $e$ intersects $P$.

(2) If the oriented plane of $g$ is identical to $P$, then the interior of $g$ is **on**$B$. If the plane of $g$ is identical to $P$, but oppositely oriented, then the interior of $g$ is **out**$B$.

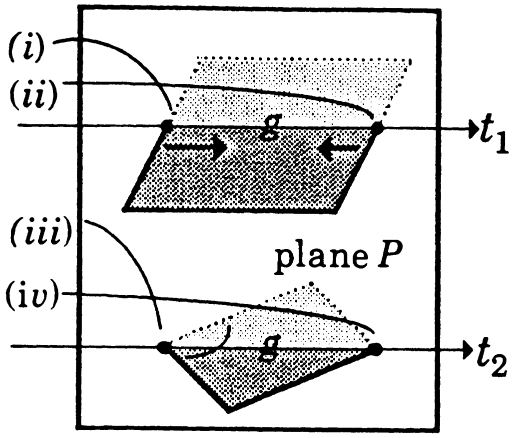
(3) Otherwise, if $g$ intersects $P$ transversely, then the intersection points computed in Step (1) are collinear. Compute a tangent $t$ along this line as $N_g \times N_P$, where $N_g$ and $N_P$ are the normals to $g$ and $P$, respectively; sort the intersection points along $t$; and remove all duplicates from the list.

(4) At each intersection point $a$, determine whether the line-segment that connects $a$ with the next point $b$ in order along the intersection line is in the interior of face $g$. This line-segment, called a *cross-face edge*, is created if $t$ points into the interior of $g$ at $a$:

   (i) If $a$ is in the interior of an edge $e$ of $g$ and $e$ has a directed edge on $g$ whose face-direction vector has a positive dot-product with $t$, then create a cross-face edge between $a$ and $b$ (see Figure 37[a]).

   (ii) If $a$ is at a vertex $v$ of $g$ and either $v$ is isolated in $g$ or $t$ splits an area-enclosing directed-edge pair of $v$ on $g$, then create a cross-face edge between $a$ and $b$ (see Figure 37[b]).

**Algorithm 6. Intersecting the faces of solid $B$ with oriented plane $P$**




(a) The Face-direction vector at $(i)$ has a positive dot-product with $t_1$, but the face-direction vector at $(ii)$ has not.

(b) Vector $t_2$ splits an area enclosing pair of directed edges at $(iii)$, but not at $(iv)$.

**Figure 37. Intersecting faces of solid $B$ with plane $P$**

For each face $g$ of $B$, Steps (1) and (2) of Algorithm 6 on page 52 examine every directed edge and vertex of $g$, and Step (3) might have to sort $O(E_g + V_g)$ points. Step (4 [ii]) requires that we insert $t$ into a radially-ordered list of vectors, corresponding to the directed edges of $g$ adjacent to a vertex of $g$. Thus Step (4) requires time $O(D_g)$. In total, for all faces of $B$, Algorithm 6 on page 52 requires time

$$O\left(\sum_{g}^{Faces(B)} D_g + [E_g + V_g] \log[E_g + V_g]\right) = O(D_B \log D_B)$$

to intersect the faces of $B$ with $P$. If the number of intersections of $P$ with each face of $B$ is bounded by a constant, then the $O(D_B \log D_B)$ time bound is reduced to $O(D_B)$.

The next stage in the construction of $G_P$ is an orientation of its edges, so that when traversing a directed edge of $G_P$, the region of $B$ to the right (on oriented plane $P$) is either **in**$B$ or **on**$B$. Then the directed edges of $G_P$ are linked to form clockwise cycles (the algorithm was described in "Chapter 2. Representing Solids"). A cross-face edge of $G_P$ gets a single directed edge, oriented by the vector $t$ of Algorithm 6. An edge of $G_P$ induced by edge $e$ of $B$ gets zero, one or two directed edges using the following idea: If on one side of $e$, oriented plane $P$ is either **in**$B$ or **on**$B$, a directed edge is created (see Algorithm 7 on page 54).

The edges of $G_P$ get zero, one, or two directed edges as follows:

(1) A cross-face edge of $G_P$ gets a single directed edge, oriented by the vector $t$ of Algorithm 6 on page 52 (see Figure 38[a]).

(2) An edge of $G_P$ induced by edge $e$ of $B$ gets zero, one, or two directed edges. First, construct face direction vectors $fd_l$ and $fd_r$ for the left and and right sides, respectively of $e$ on $P$ (recall Figure 16 on page 20) Then see which of $fd_l$ and $fd_r$ either point into faces of $B$ that are **on**$B$ or split volume-enclosing face-pairs of $e$ (point into **in**$B$ sets). There are three cases:

   (i) if neither $fd_l$ nor $fd_r$ points into a subset of $P$ that is **in**$B$ (or **on**$B$) then $G_P$ gets only the vertices of $e$, and the edge of $G_P$ is removed (see Figure 38[b]);

   (ii) if both $fd_l$ and $fd_r$ point into subsets of $P$ that are **in**$B$ (or **on**$B$) then create two oppositely-oriented directed edges (see Figure 38[c]); finally,

   (iii) if only one of $fd_l$ and $fd_r$ points into a subset of $P$ that is **in**$B$ (or **on**$B$) then create a single directed edge on $G_P$, oriented with the **in**$B$ (or **on**$B$) subset of $P$ to the right (see Figure 38[d]).

**Algorithm 7. Orienting edges of $G_P$**

In order to get a bound on the running time of Algorithm 7 we observe that plane $P$ might intersect each edge or vertex of $B$, and so the number of vertices of $G_P$ is bounded by $E_B + V_B \leq 2E_B$. Furthermore, $G_P$ is a planar graph, and so $E_{G_P} \leq 3V_{G_P}$ and $E_{G_P} \leq 6E_B$. Thus Algorithm 7 orients the cross-face edges of $G_P$ in time $O(E_{G_P}) = O(E_B)$. In order to orient an edge of $G_P$ induced by edge $e$ of $B$, we must determine if a face-direction vector $fd$ splits a volume-enclosing face-pair of $e$. We do this by inserting $fd$ into a radially-ordered list of face-direction vectors that correspond to the directed edges associated with $e$. Used to orient all edges of $G_P$, Algorithm 7 has an asymptotic running time of $O(E_{G_P} + D_B) = O(D_B)$, because every directed edge of $B$ might have to be examined in order to orient the edges of $G_P$. If the number of faces defining each edge of $B$ is bounded by a constant, Algorithm 7 will orient the edges of $G_P$ in time $O(D_{G_P}) = O(E_B)$.

As Algorithm 7 creates the directed edges of $G_P$, we construct lists of directed edges of $G_P$ incident to each vertex of $G_P$. We also construct lists of directed edges associated with each edge of $G_P$. The time needed to construct these lists is $O(D_{G_P}) = O(E_B)$.

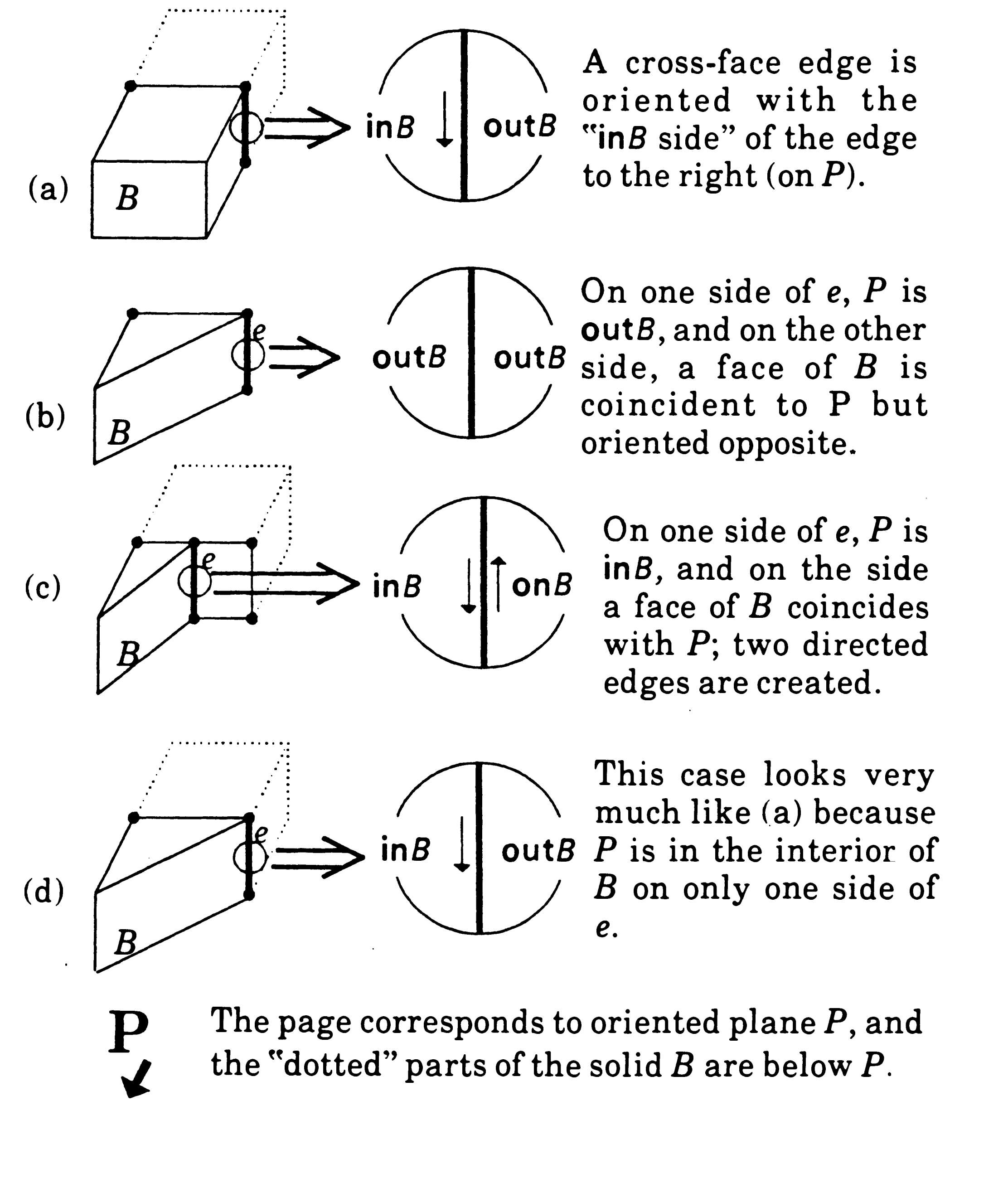



**Figure 38. Orienting edges of $G_P$**

It is possible that $G_P$ has *isolated vertices*, which are defined to be vertices of $G_P$ that are not incident to any edges of $G_P$. We need to determine which isolated vertices of $G_P$ are contained in **on**$B$, **in**$B$, and **out**$B$ subsets of $P$. Because each isolated vertex of $G_P$ is induced by a vertex of $B$, we can distinguish the three cases by examining edges and faces of $B$ incident to the inducing vertex of $B$. A vertex isolated in an **on**$B$ subset of $P$ is easy to distinguish but the second two cases are harder (see Algorithm 8 on page 56).

Given a vertex $v$ of $B$ that induces an isolated vertex of $G_P$, we determine whether the isolated vertex of $G_P$ is contained in an **in**$B$, **on**B, or **out**$B$ subset of $P$ (see also Figure 39).

(1) If $v$ is isolated in a face of $B$ that is **on**$B$, then $v$ is located in an **on**$B$ subset of $P$.

(2) Find the edge $e$ of $B$ incident to $v$ that forms the shallowest angle with $P$. If $e$ lies in $P$ then $v$ is contained in an **out**$B$ subset of $P$ (by Algorithm 7 on page 54).

(3) Construct a plane $Q$ that is orthogonal to $P$ and contains $e$.

(4) Construct the face direction vector $fd$ on $Q$ from $e$, pointing towards $P$.

(5) Vertex $v$ is contained in an **in**$B$ subset of $P$ if $fd$ splits a volume-enclosing pair of faces adjacent to $e$. Otherwise $v$ is contained in an **out**$B$ subset of $P$.

**Algorithm 8. Isolated vertices of $G_P$ are in in$B$, on$B$, or out$B$ subsets of $P$**

**Lemma 7.** Algorithm 8 determines whether an isolated vertex of $P$ induced by vertex $v$ of $B$ is in an **in**$B$, **on**$B$, or **out**$B$ subset of $P$.

**Proof.** The neighbourhoods found by Steps (1) and (2) are straightforward. Otherwise, plane $Q$ is uniquely defined (to sign) by $e$ and $P$, because $e$ cannot be orthogonal to $P$. Finally, $fd$ cannot lie on a face adjacent to $e$, and hence $fd$ classifies the neighbourhood of $v$ on $P$. ■

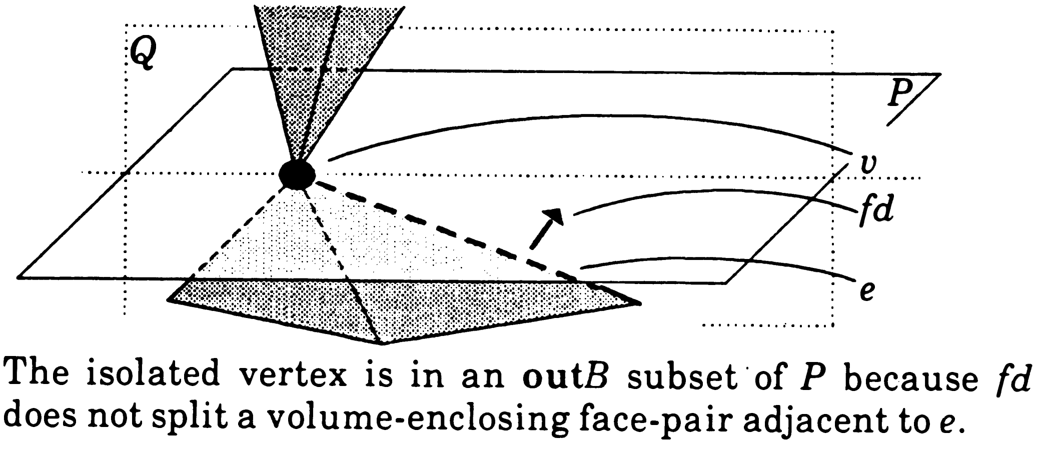



**Figure 39. Isolated vertices of $G_P$ are in in$B$, on$B$, or out$B$ subsets of $P$**

Step (2) of Algorithm 8 examines each directed edge adjacent to the isolated vertex $v$ and Step (5) examines every directed edge of one edge $e$ adjacent to $v$.

Because the number $D_e$ of directed edges of $e$ is bounded by the number of directed edges $D_v$ adjacent to $v$, Algorithm 8 requires time $O(D_v)$ to determine whether $v$ is **in**$B$, **on**$B$, or **out**$B$. The total number of vertices of $B$ that might be isolated on $P$ is bounded by the number $D_B$ of vertices of B, and because each directed edge of $B$ is adjacent to two vertices, the total time required by Algorithm 8 to examine them is $O(D_B)$.

The last stage in the construction of $G_P$ is the organization of its directed edges into bounding cycles. Recall from "Chapter 2. Representing Solids" that directed-edge cycles of a face are induced by a radial ordering of directed edges around incident vertices on the face plane. Thus, the directed edges of $G_P$ must be radially ordered around the vertices on $P$ (see Appendix C). The time required to radially order these directed edges is

$$O\left( \sum_{v}^{Vertices(G_P)} E_v \log E_v \right) = O(E_{G_P} \log E_{G_P}).$$

Recall that $E_{G_P} \leq 6E_B$, and so the above expression is $O(E_B \log E_B)$.

The construction of $G_P$ is repeated once for each face of solid $A$, and so it requires time

$$O(F_A[D_B \log D_B + E_B \log E_B]) = O(F_A D_B \log D_B)$$

to construct all cross-sections of $B$. If every vertex of every $G_P$ has constant degree and the planes of the faces of $A$ each intersect a constant number of edges and vertices of each face of $B$, the constructions require total time $O(F_A D_B)$.

## *4.4 Intersecting Faces of Solid A with Cross-Sections of Solid B*

After computing the cross-sections of solid $B$ using the oriented planes of faces of solid $A$, we intersect each face $f$ of $A$ with subsets of the corresponding cross-section. More precisely, given a face $f$ of $A$ with oriented plane $P$, we intersect $f$ with the subsets of $G_P$ that are **in**$B$ and **on**$B$ (recall that $G_P$ is the cross-section of $B$ with $P$). The results of this intersection are connected regular sets on $P$, each of which is a face of $A \cap^* B$. First we will intersect the edges and vertices of $f$ with the edges and vertices of $G_P$ and then we will trace out the boundaries of the connected regular sets that are in $f$, and in **in**$B$ and **on**$B$ subsets of $P$ (see Figure 33[b]and Figure 36 on page 52).

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

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

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

(4) Add directed edge cycles and isolated vertices of $f$ (respectively $G_P$) contained in $f$ (resp. **in**$B$ and **on**$B$ subsets of $G_P$).

**Algorithm 9. Regularized intersection of $f$ and $G_P$**

In order to guarantee that the faces of $A \cap^* B$ constructed by Algorithm 9 are regular sets, some of the intersection points found in Step (1) of Algorithm 9 are discarded. Before seeing how to do this, we will describe how to intersect the directed edges and vertices of $f$ with those of $G_P$ in more detail (see Algorithm 10).

Given a face $f$ of solid $A$ and the corresponding cross-section $G_P$, the edges and vertices of $f$ are intersected with the edges and vertices of $G_P$ as follows:

(1) For each isolated vertex $v$ of $f$, determine if any vertices or edges of $G_P$ coincide with $v$.

(2) For each edge $e$ of $f$, intersect $e$ with the edges and vertices of $G_P$.

(3) For each edge $e$ of $f$ sort, along $e$, the intersections of $e$ with the edges and vertices of $G_P$.

(4) For each edge $e'$ of $G_P$ sort, along $e'$, the intersections of $e'$ with the edges and vertices of $f$.

**Algorithm 10. Intersecting the edges and vertices of $f$ and $G_P$**

If in Step (2) of Algorithm 10, we discover that an edge of $G_P$ overlaps an edge of $f$, then the intersection of the vertices of one edge with the other edge (and vice versa) constitute the intersection points. All intersecting edges are subdivided by their intersection points. For example, two edges intersecting in interior points become four edge segments, and directed-edge segments inherit orientations.

The time required by Algorithm 10 is bounded by the time required to intersect the edges and vertices of $f$ and $G_P$ and then sort those intersection points along the intersecting edges. If there are $k$ intersection points and $n$ lines, the algorithm of Chazelle and Edelsbrunner [1988] can implement Algorithm 10 in time $O(n \log n + k)$. However, a simpler algorithm that just intersects all possible pairs of edges and vertices is implemented: Steps (1-2) of Algorithm 10 on page 58 first obtain the edges and vertices of $f$ and $G_P$ from their directed edge cycles and isolated vertices. Then these edges and vertices are intersected. Because there are at most $E_f + V_f$ and $E_{G_P} + V_{G_P}$ edges and vertices in $f$ and $G_P$, respectively, Steps (1-2) require time

$$O(D_f + D_{G_P} + [E_{G_P} + V_{G_P}][E_f + V_f])$$

The running time of Step (3) is determined by the number of intersection points contained in each edge of $f$. This is bounded by $E_{G_P} + V_{G_P}$, which is in turn bounded by number $E_B$ of edges of $B$. Thus Step (3) requires time $O(E_f E_B \log E_B)$. Similarly, Step (4) requires time $O(E_B[E_f + V_f] \log[E_f + V_f])$ to sort the intersection points along the edges of $G_P$.

In total, to intersect edges and vertices of every face of $A$ with the edges and vertices of the corresponding $G_P$, Steps (1-2) of Algorithm 10 require time

$$\begin{aligned} T_{(1,2)} &= O\left( \sum_{f}^{Faces(A)} D_f + D_{G_P} + [E_{G_P} + V_{G_P}][E_f + V_f] \right) \\ &= O\left( D_A + F_A E_B + \sum_{f}^{Faces(A)} E_B[E_f + V_f] \right) \\ &= O(D_A + F_A E_B + D_A E_B) \\ &= O(E_A[F_A + D_A]). \end{aligned}$$

In total, Step (3) sorts the intersection points found in Steps (1-2) along the edges of every face of $A$ in time

$$T_{(3)} = O(E_A E_B \log E_B),$$

and Step (4) sorts the points along every edge of every $G_P$ in total time

$$
\begin{aligned}
T_{(4)} &= \mathrm{O}\left( E_B \sum_{f}^{Faces(A)} [E_f + V_f] \log[E_f + V_f] \right) \\
&= \mathrm{O}\left( E_B \log D_A \sum_{f}^{Faces(A)} E_f + V_f \right) \\
&= \mathrm{O}(E_B D_A \log D_A).
\end{aligned}
$$

Summarizing, the time required by Algorithm 10 to intersect the edges and vertices of $A$ with those of the cross-sections of $B$, and then sort the intersection points along the intersecting edges is bounded by

$$
\begin{aligned}
T_{(1,2)} + T_{(3)} + T_{(4)} &= \mathrm{O}(E_A E_B \log E_B + E_B D_A \log D_A) \\
&= \mathrm{O}(D_A E_B \log[D_A E_B]).
\end{aligned}
$$

The asymmetry in the above expression reflects the fact that the $G_P$'s are computed independently. If Steps (3) and (4) sort only a constant number of intersection points along every edge, then Algorithm 10 requires time $\mathrm{O}(D_A E_B)$.

Once the edges and vertices of a face $f$ of $A$ have been intersected with those of the corresponding $G_P$ to obtain intersection points, these points are analyzed. At each intersection point we determine which edge segments of $G_P$ (resp. $f$) are contained in the interior of $f$ (resp. **in**$B$ or **on**$B$ subsets of $G_P$); we use this containment information to discard certain intersection points and hence regularize the intersection.

As a first step in the intersection-point analysis, directed edge-segments incident to each intersection point are radially merged. This merge is effected by radially merging vectors parallel to the directed edge-segments and oriented away from the intersection point. For example, if vertex $v$ of $G_P$ is contained by an edge $e$ of $A$ that has two directed edges on $f$, then the vectors parallel to each directed-edge of $G_P$ and oriented away from $v$ are radially merged with four vectors parallel to the intersected edge of $f$ (two oppositely oriented vectors for each directed edge of $e$ on $f$). A case analysis describing this merge process is given in Algorithm 11 on page 61.

The directed edge-segments of $f$ and $G_P$ incident to intersection point $a$ are merged by radially merging two sets of vectors, one set parallel to the edges of $f$ incident to $a$ and the other set parallel to the edges of $G_P$ incident to $a$.

If $a$ is in the interior of an edge $e$ of $f$ ( resp. $G_P$), then the set of vectors parallel to $e$ is obtained by taking, for each directed edge of $e$ on $f$ ( resp. $G_P$), two oppositely-oriented vectors parallel to $e$.

If $a$ coincides with a vertex $v$ of $f$ ( resp. $G_P$), then the set of vectors parallel to the edges of $f$ ( resp. $G_P$), is obtained by taking, for each directed edge $\vec{e}$ of $f$ ( resp. $G_P$) adjacent to $v$, a vector parallel to $\vec{e}$ and oriented away from $v$.

The possible cases are as follows:

(i) $a$ is in the interior of an edge of $f$ and $G_P$ (Figure 40[a]);

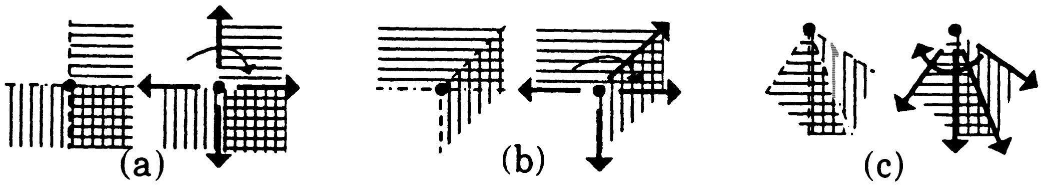
(ii) $a$ is a vertex of $G_P$ and is in the interior of an edge of $f$ (Figure 40[b]);

(iii) $a$ is a vertex of $f$ and is in the interior of an edge of $G_P$ (Figure 40[b]);

(iv) $a$ is a vertex of both $f$ and $G_P$ (Figure 40[c]); and finally,

(v) in cases (ii-iv), $a$ can also be an isolated vertex of either $f$ or $G_P$, and then no merge is necessary (not shown in Figure 40).

**Algorithm 11. Local analysis at intersection points**




Directed edge-segments are radially sorted around an edge/edge intersection (a), an edge/vertex intersection (b), and a vertex/vertex intersection (c), and intersecting areas in the neighbourhood of the intersections are marked.

**Figure 40. Local analysis at intersection points**

The radially merged list of directed edge-segments incident to an intersection point produced by Algorithm 11 is examined twice. In the first pass, all directed edge-segments of $f$ that lie within **in**$B$ or **on**$B$ subsets of $G_P$ are marked; in the second pass, all directed edge-segments of $G_P$ that lie within $f$ are marked. All unmarked directed edge-segments are then discarded. Coincident identically-oriented 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 $A \cap^* B$.

Algorithm 11 has a running time that is linear in the number of directed edge-segments incident to the intersection point. Each directed edge-segment is incident to at most two intersection points and so is merged by Algorithm 11 at most twice in the intersection-point analysis of $f \cap G_P$. Thus Algorithm 11 requires time $O(D_f + D_{G_P} + V_f + V_{G_P} + D_f D_{G_P})$ to analyze all of the intersection points of $f \cap G_P$. For all faces of $A$ Algorithm 11 requires total time

$$O\left( \sum_{f}^{Faces(A)} D_f + D_{G_P} + V_f + V_{G_P} + D_f D_{G_P} \right) = O(D_A E_B).$$

After the intersection-point analysis is complete, the list of radially-merged directed-edge segments incident to the point is saved so that bounding directed-edge cycles of faces of $A \cap^* B$ on the plane $P$ of $f$ can be traced out by traversing directed edges of $f$ and $G_P$ between intersection points. This traversal requires time that is linear in the number of directed-edge segments of $f \cap G_P$, which (as above) is

$$O(D_f + D_{G_P} + V_f + V_{G_P} + D_f D_{G_P});$$

and for all faces of $A$ and corresponding $G_P$ the traversal requires $O(D_A E_B)$.

After directed-edge cycles and isolated vertices that are induced by intersections of edges and vertices of $f$ and $G_P$ have been constructed, there may remain cycles or isolated vertices of $G_P$ contained in $f$ or cycles or isolated vertices of $f$ contained in **in**$B$ or **on**$B$ subsets $G_P$. For example, consider the cycles of Figure 41 on page 63. Here the face $f$ is bounded by $C_1$ and $C_2$, and $G_P$ contains two **in**$B$ (or **on**$B$) subsets, 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 subset of $G_P$ and is also discarded. As result, the intersection area is bounded by the cycles (1,2,3,4), $C_4$, and $C_5$. This cycle containment is tested by casting a ray across $P$ (see Algorithm 12 on page 63).

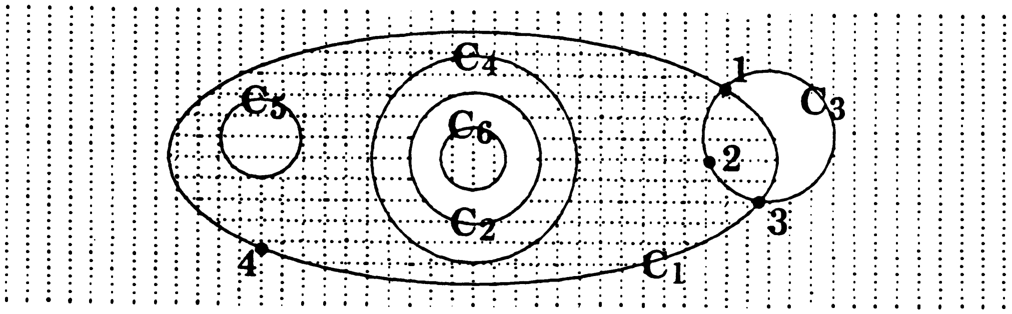



**Figure 41. Intersecting $f$ and $G_P$**

To test for the containment of a cycle (or isolated vertex) $C$ by the area bounded by group **C** of cycles and isolated vertices, perform the following steps:

(1) Pick a point $u$ on $C$ and a point $v$ on one of the cycles of **C**.

(2) Intersect the line containing $u$ and $v$ with the cycles of **C**, partitioning the line into segments that are inside or outside the area bounded by the cycles of **C** (see Figure 42).

(3) Cycle $C$ is contained by the area bounded by **C** if the segment containing $u$ is in the area bounded by **C**. This is determined by a local analysis like Cases (i) or (ii) of Algorithm 6 on page 52.

**Algorithm 12. Cycle Containment Test**

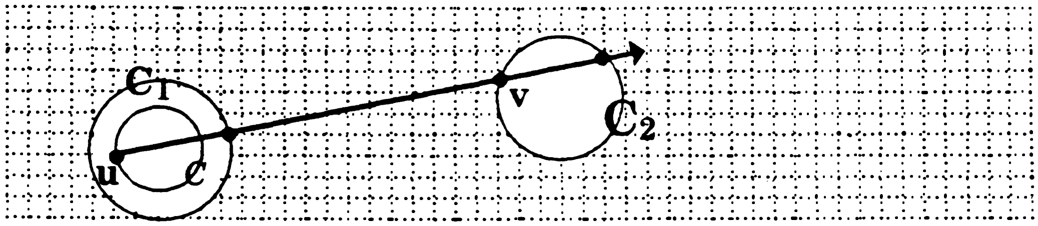
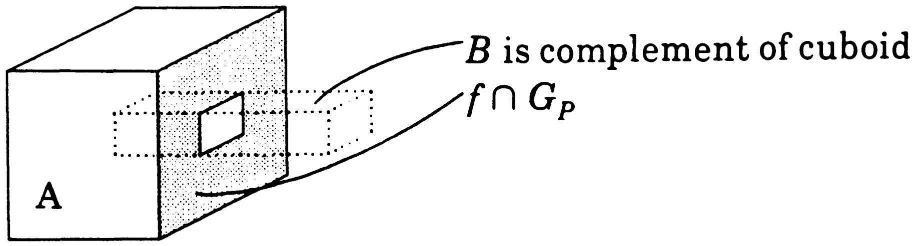



**Figure 42. Cycle Containment Test**

The running time of Algorithm 12 is determined by the time needed to execute Step (2), which is linear in the number of edges and vertices in the group **C** of directed edge cycles and isolated vertices. For a face $f$ of $A$, this number is $D_f + V_f$, and for the corresponding $G_P$ this number is $D_{G_P} + V_{G_P}$, which is $O(E_B)$. Algorithm 12 tests for containment of a cycle or isolated vertex of $f$ inside **in**$B$ or **on**$B$ subsets of $G_P$ in time $O(D_B)$, and tests for containment of a cycle or isolated vertex of $G_P$ in the interior of $f$ in time $O(D_f + V_f)$. Potentially, Algorithm 12 could be used to test the containment of every cycle and isolated vertex of every face of $A$ (and its corresponding $G_P$). There are potentially $O(E_f)$ directed edge cycles and $O(V_f)$ isolated vertices of $f$ and $O(E_B)$ directed edge cycles and isolated vertices of $G_P$, and so the total time required by Algorithm 12 to test them for containment is

$$O\left( \sum_{f}^{Faces(A)} E_B[E_f + V_f] \right) = O(D_A E_B).$$

Finally, it is possible that a face of solid $A$ is in $A \bigcap^* B$ even though the corresponding cross-section $G_P$ contains no edges or vertices. 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 $A \bigcap^* B$ because it is reachable by vertex and edge adjacent faces from another face $f$ of $A$ that intersects the surface of $B$ (see Figure 43). Thus faces of $A$ that are in the interior of $B$ are transitively added to $A \bigcap^* B$ by traversing the boundary of $A$. This process requires time $O(|A|)$.




Face $f$ is the only face of $A$ that induces a cross-sectional graph. The other faces of $A \cap^* B$ in the interior of $A$ are those transitively adjacent to edges of $f$.

**Figure 43. Adding faces of $A$ in the interior of $B$**

Summarizing this section, Steps (2) and (3) of Algorithm 5 on page 49 (intersecting faces of A with corresponding cross-sections of $B$ and constructing faces of $A \cap^* B$ on the surface of $A$) require time $O(D_A E_B \log[D_A E_B])$. This bound is reduced to $O(D_A E_B)$ if no edges of $A$ intersect more than a constant number of faces of $B$ (the log terms arise from sorting intersecting points).

## 4.5 Intersecting Faces of Solid B with Solid A

When the processing of the previous section is complete, the boundaries of faces of $A \cap^* B$ on the surface of $A$ and in the interior of $B$ that result from intersecting shells of $A$ and $B$ have been found. The next steps in the intersection algorithm (Steps [4-5] of Algorithm 5 on page 49) construct bounding directed edge cycles and isolated vertices of faces of $A \cap^* B$ that are in the interior of $A$ and on the surface of $B$. The simplest way to obtain these faces is to run Steps (1-3) of Algorithm 5 with the roles of $A$ and $B$ interchanged, but a more direct approach is used.

We examine the edges and vertices of $A \cap^* B$ to find those which lie on the surface of $B$, *transfer* those edges and vertices to faces of $B$ extending inside $A$, and then use the transferred edges and vertices to intersect the faces of $B$ with the interior of $A$ (see also Figure 34 on page 48). Then the boundaries of faces of $A \cap^* B$ on the surface of $B$ and in the interior of $A$ are traced out. Areas that lie on the surfaces of both $A$ and $B$ can be ignored, as they were classified as **on**$B$ and have already been accounted for.

We first transfer edges of $A \cap^* B$ to faces of $B$, orienting the edges on the faces of $B$ appropriately (see Algorithm 13 on page 66).

Given oriented plane $P$ of a face $f$ of solid $A$, and given cross-section $G_P$ of $B$ with $P$, edges of $A \cap^* B$ are transferred to faces of $B$ in the following manner:

(i) A segment of an edge of $f$ located in an **in**$B$ area of $G_P$ is not transferred (see Figure 44[a]).

(ii) A segment of edge $e$ of $f$ located in an **on**$B$ area of $G_P$ is transferred to the corresponding face of $B$ if $e$ has exactly one directed edge on $f$. The orientation of the transferred edge segment is opposite that of the directed edge of $e$ on $f$ (see Figure 44[b]).

(iii) A segment of a cross-face edge $\bar{e}$ of $G_P$ located in the interior of $f$ is transferred to the face $g$ of $B$ that induced $\bar{e}$. The orientation of the transferred edge segment is opposite that of the directed edge of $\bar{e}$ on $G_P$ (see Figure 44[c]).

(iv) A segment of an edge $e'$ of $B$ located in the interior of $f$ is transferred to each face of $B$ adjacent to $e'$ and below $f$. A face $g$ adjacent to $e'$ is below $f$ if a face-direction vector that points from $e'$ into the interior of $g$ also points below $f$ (see Figure 44[d]). The orientations of the transferred edge segments are given by the directed edges of $e'$.

(v) A segment of a cross-face edge $\bar{e}$ of of $G_P$ coincident with an edge $e$ of $f$ is transferred to the face $g$ of $B$ that induced $\bar{e}$ if the face-direction vector that points from $\bar{e}$ into the interior of $g$ splits a volume-enclosing pair of faces adjacent to $e$ (not shown in Figure 44). The orientation of the transferred edge segment is opposite that of the directed edge of $\bar{e}$ on $G_P$.

(vi) A segment of an edge $e'$ of $B$ that coincides with an edge $e$ of $f$ is transferred to each face of $B$ adjacent to $e'$ that splits a volume-enclosing pair of faces adjacent to $e$. This determination is made by merging the radially-ordered lists of face-direction vectors of $e$ and $e'$ (see Figure 44[e]). The orientations of the transferred edges are given by the directed edges of $e'$.

**Algorithm 13. Transferring edges of $A \cap^* B$ to faces of $B$**

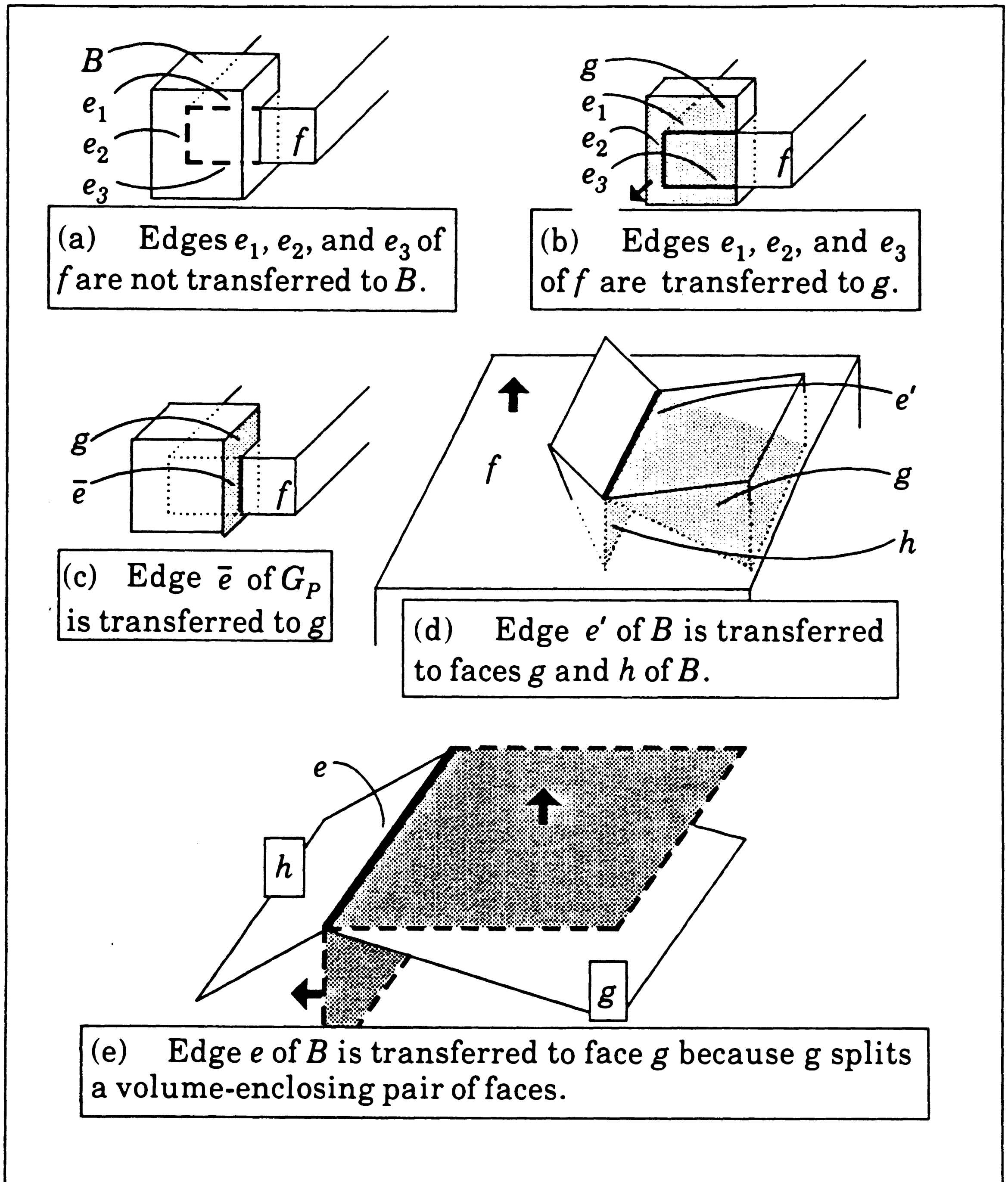



**Figure 44. Transferring edges of $A \cap^* B$ to faces of $B$**

The transference of an edge of $A \cap^* B$ to a face of $B$ can be done in constant time if the edge does not coincide with an edge of either $A$ or $B$ (Case [iii] of Algorithm 13). If the edge of $A \cap^* B$ coincides with an edge $e'$ of $B$ then every directed edge of $e'$ is examined (Cases [iv] and [vi]). If the edge of $A \cap^* B$ coincides with an edge $e$ of $f$ then every directed edge of $e$ is examined (Cases [ii] and [v] of Algorithm 13). In order to bound the time needed by Algorithm 13 to transfer every edge of $A \bigcap^* B$ to the appropriate face(s) of $B$, we analyze each case separately.

(ii) Edge $e$ of $A$ might intersect $O(E_B)$ edges of $B$, and so the total time required is

$$O(E_A E_B).$$

(iii) For each face $f$ of $A$, there might be $O(E_B)$ cross-face edges that intersect every edge and vertex of of $f$, and so the transference of every segment of every cross-face edge requires time

$$O\left( E_B \sum_{f}^{Faces(A)} E_f + V_f \right) = O(D_A E_B).$$

(iv) As in Case (iii), edge $e'$ might intersect every edge of $A$, and so the transference of every segment of every edge of $B$ to a single face of $B$ requires time

$$O\left( \sum_{e'}^{Edges(B)} E_A D_{e'} \right) = O(E_A D_B).$$

(v) We could potentially find every cross-face edge of every $G_P$ coincident with every edge of $A$, and so the total time required to transfer cross-face edge-segments is

$$O\left( E_B \sum_{e}^{Edges(A)} D_e \right) = O(D_A E_B).$$

(vi) Finally, every edge of $B$ might be coincident with and subdivided by $O(E_A)$ edges of $A$,and so the total time to transfer all of the edge segments is

$$O\left( \sum_{e'}^{Edges(B)} \sum_{e}^{Edges(A)} D_{e'} D_e \right) = O(D_A D_B).$$

Summarizing, Algorithm 13 requires time $O(D_A D_B)$ to transfer the edges of $A \bigcap^* B$ to the appropriate faces of $B$ and then orient the transferred edges.

After edges of $A \bigcap^* B$ have been transferred to faces of $B$, it may be necessary to transfer vertices as well. Vertex $v$ of $A \bigcap^* B$ is transferred if $v$ is a non-manifold vertex of $B$ (recall Figure 10 on page 13) or if Algorithm 13 did not transfer any adjacent edges of $v$ to $B$. There are three situations: the vertex may be in a face interior, on the edge of a face, or coincident with a vertex. For example, in Figure 45(a), tetrahedron $B$ is intersected with cube $A$. The surface of $B$ is contained entirely within $A$ except for a vertex of $B$ that requires the faces of $B$ to be added to the surface of $A \cap^* B$. In Figure 45(b), the surfaces of the deformed cube ($A$) and the triangular cylinder ($B$) intersect at a point in the interior of two edges. This intersection point must be transferred to $B$ as well.


In (a), the surfaces of the cube and tetrahedron intersect at the tetrahedral apex; and in (b), the surfaces intersect at a single point (two edges intersect).

**Figure 45. Vertices of $A \cap^* B$ transferred to faces of $B$**

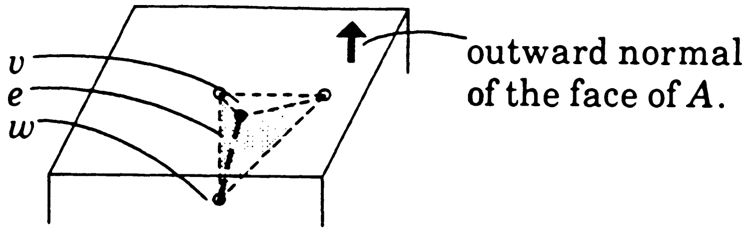
For a vertex $v$ of $A \cap^* B$ in the interior of a face of $A$, we determine which edges of $A \cap^* B$ adjacent to $v$ are inside $A$, and transfer those edges to faces of $B$ incident to $v$ (see Algorithm 14). For each edge $e$ of $B$ incident to $v$ there are $O(D_e)$ such faces, and so Algorithm 14 requires time $O(D_B)$ to transfer edges of $A \cap^* B$ incident to vertices of $B$ if those vertices of $B$ are located in the interiors of faces of $A$.

If $v$ is a vertex of $A \cap^* B$ contained in the interior of a face of $A$, and $e$ is an edge of $A \cap^* B$ adjacent to $v$ and contained by an edge of $B$, then $e$ is transferred to the faces of $B$ containing $e$ as follows:

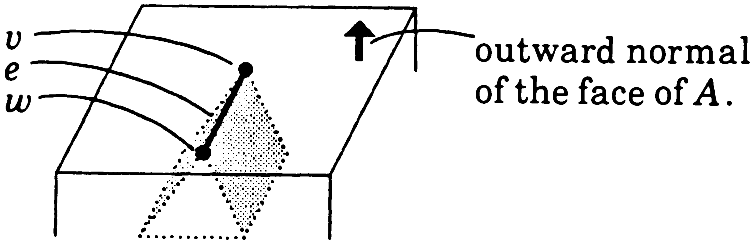
(1) if the other endpoint of $e$ is below the face of $A$, then transfer $e$ to its adjacent faces (see Figure 46[a]); otherwise,

(2) if $e$ lies in the plane of the face of $A$, then transfer $e$ to each of its adjacent faces that are below the plane (using Case [iv] of Algorithm 13 on page 66; see also Figure 46[b]).

**Algorithm 14. Transferring a vertex in a face of $A$**




(a) Vertex $v$ is in the face of $A$ and the other endpoint $w$ of $e$ is below the face of $A$, and so $e$ is transferred to its adjacent faces.




(b) Vertex $v$ is in the face of $A$ and the other endpoint $w$ of $e$ is also in the face, and so $e$ is transferred to its adjacent faces that are below the face.

**Figure 46. Transferring a vertex in a face of $A$**

For a vertex $v$ of $A \cap^* B$ in the interior of an edge of $A$, we determine which edges of $A \cap^* B$ adjacent to $v$ split volume enclosing face-pairs adjacent to the edge of $A$, and transfer those edges to faces of $B$ incident to $v$ (see Algorithm 15 on page 71).

If $v$ is a vertex of $A \cap^* B$ contained in the interior of an edge of $A$, and $e$ is an edge of $A \cap^* B$ adjacent to $v$ and contained by an edge of $B$, then $e$ is transferred to the faces of $B$ containing $e$ as follows:

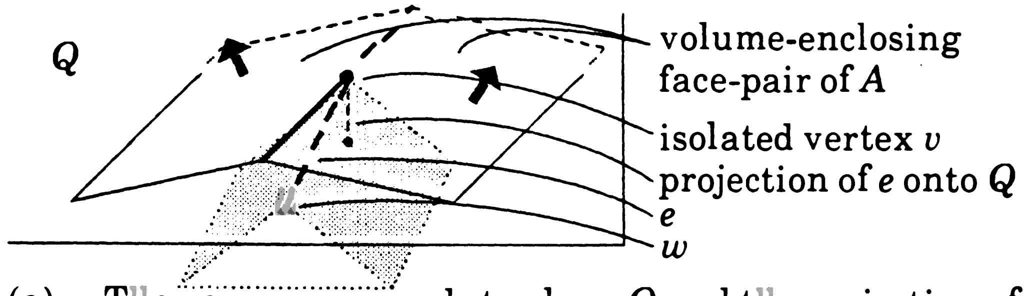
(1) Project the other endpoint of $e$ onto the plane orthogonal to the edge of $A$ and then determine if $e$ splits a volume-enclosing pair of faces adjacent to the edge of $A$ by examining the radially-sorted face-direction vectors associated with the edge of $A$;

(2) if $e$ splits a volume-enclosing face-pair of the edge of $A$, then transfer $e$ to the faces of $B$ that contains $e$ (see Figure 47[a]);

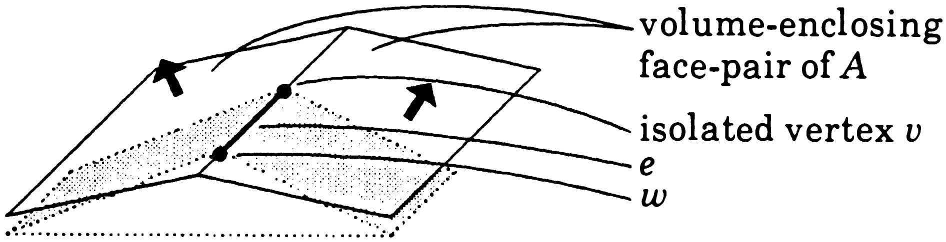
(3) if $e$ coincides with the edge of $A$, then use Case [vi] of Algorithm 13 on page 66 to transfer $e$ to the faces of $B$ that contain $e$ and split volume-enclosing face-pairs of the edge of $A$ (see Figure 47[b]); and

(4) if $e$ lies on a face of $A$, then use Case [iv] of Algorithm 13 to transfer $e$ to faces of $B$ that contain $e$ and are below the face of $A$.

**Algorithm 15. Transferring a vertex in an edge of $A$**




(a) The page corresponds to plane $Q$ and the projection of $e$ onto $Q$ splits a volume-enclosing pair of faces adjacent to the edge of $A$, and so $e$ is transferred to its adjacent faces.




(b) Edge $e$ coincides with an edge of $A$, and so $e$ is transferred to each of its adjacent faces that split a volume-enclosing pair of faces of the edge *of* $A$.

**Figure 47. Transferring a vertex in an edge of *A***

A bound on the total time required by Algorithm 15 is obtained by observing that the algorithm examines the directed edges of $B$ incident to vertex $v$ of $A \cap^* B$ once, and for every edge of $B$ incident to $v$, the algorithm examines every directed edge of the containing edge of $A$ once. The total time required by Algorithm 15 to transfer edges of $A \cap^* B$ incident to vertices of $B$ is

$$O\left( \sum_{v}^{Vertices(B)} D_v + E_v D_A \right) = O(D_A D_B);$$

and the total time required by Algorithm 15 to transfer edges of $A \cap^* B$ adjacent to vertices of $A \cap^* B$ that are contained by edges of $B$ is.

$$O\left( \sum_{e}^{Edges(B)} D_e + D_A \right) = O(D_A E_B).$$

For a vertex $v$ of $A \cap^* B$ coincident with a vertex of $A$, we determine which edges of $B$ incident to $v$ are in the interior of $A$ by examining cross-sections of the neighbourhood of the vertex of $A$ (see Algorithm 16). Then segments of these edges of $B$ are transferred to adjacent faces of $B$.

If $v$ is a vertex of $A \cap^* B$ coincident with a vertex of $A$, and $e$ is an edge of $A \cap^* B$ adjacent to $v$ and contained by an edge of $B$, then $e$ is transferred to the faces of $B$ containing $e$ as follows:

(1) Construct a plane $Q$ that contains both $e$ and an edge of $A$ incident to $v$ (Figure 48[a]).

(2) Intersect $Q$ with the edges and faces of $A$ incident to $v$. The result of the intersection determines a collection of sectors on $Q$ centred at $v$. Each sector is either inside $A$, on some face of $A$, or outside $A$. The sector boundaries are defined either by edges of $A$ that lie in $Q$ or by intersections of faces of $A$ with $Q$. Because $Q$ contains $e$, $e$ is either in a sector or coincident with a sector boundary (see Figure 48[b]).

(3) Once we have constructed the sectors on $Q$, we can find which sector contains $e$ as follows: Define a *directed sector boundary* to be a unit vector parallel to a sector boundary and directed away from $v$. Find the largest dot product between a directed sector boundary and the unit vector parallel to $e$ directed away from $v$. This determines the closest sector boundary.

(4) If $e$ coincides with the closest sector boundary, then:

   (i) if this closest sector boundary is the intersection of $Q$ with a face of $A$, then use Case [iv] of Algorithm 13 on page 66 to transfer $e$ to faces of $B$ that contain $e$ and are below the face of $A$; otherwise,

   (ii) this closest sector boundary is an edge of $A$, and so Case [vi] of Algorithm 13 is used to transfer $e$ to faces of $B$ that contain $e$ and split volume-enclosing face-pairs of the edge of $A$.

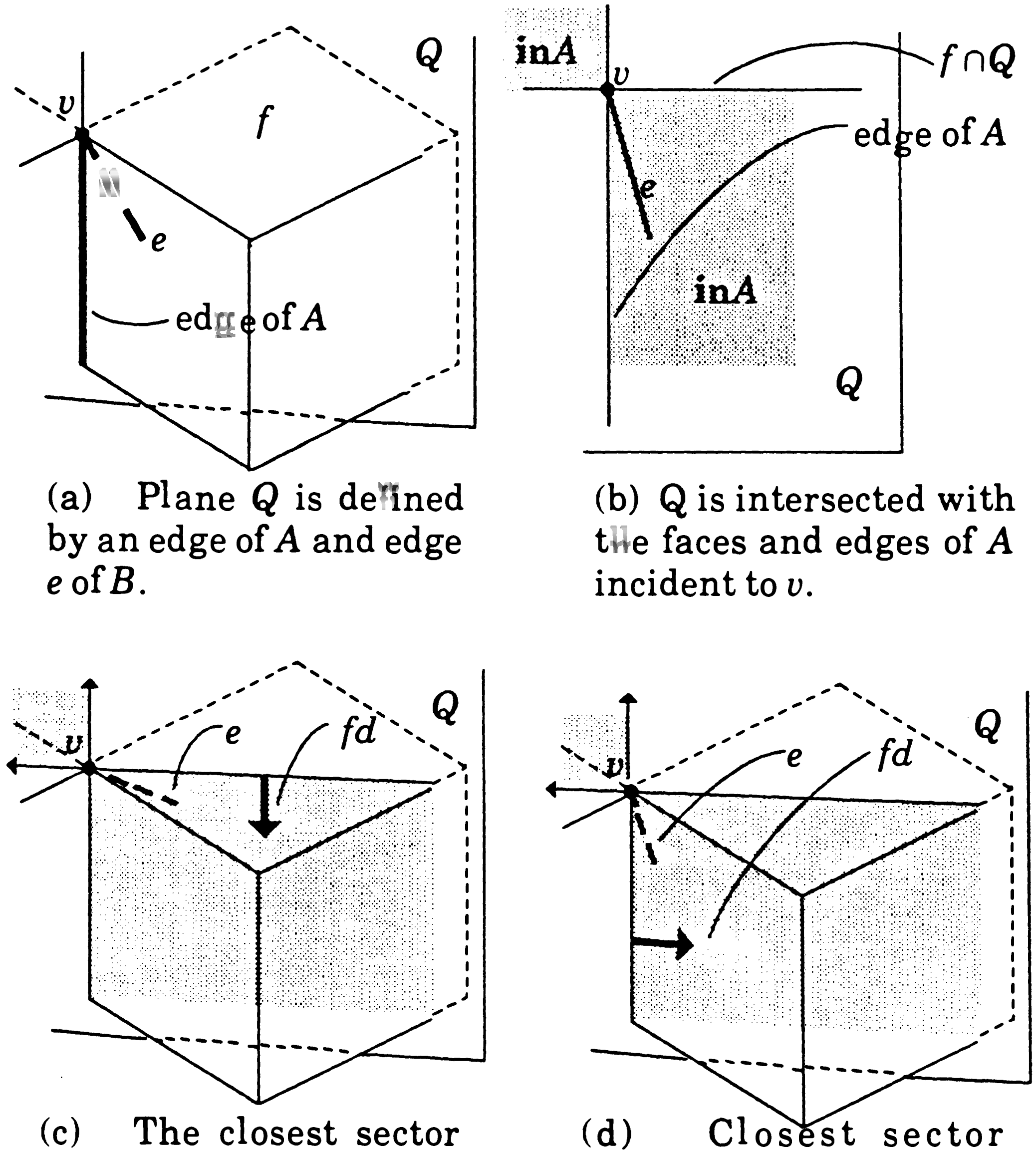
(5) Otherwise, construct the vector $fd$ that is the cross product of the normal of $Q$ with the closest directed sector boundary. Adjust the sign of $fd$ so that $fd$ points from the interior of the closest sector boundary to the interior of the sector containing $e$.

(6) If the sector containing $e$ is on a face of $A$, then use Case [iv] of Algorithm 13 on page 66 to transfer $e$ to faces of $B$ that contain $e$ and are below the face of $A$.

(7) If the closest sector boundary is the intersection of $Q$ with a face of $A$, then transfer $e$ to the faces of $B$ that contain $e$ if $fd$ points below the face of $A$ (see Figure 48[c]).

(8) Otherwise, the closest sector boundary is an edge of $A$. If $fd$ splits a volume-enclosing pair of faces adjacent to the edge of $A$, then transfer $e$ to the adjacent faces of $B$ that contain $e$ (see Figure 48[d]).

**Algorithm 16. Transferring a vertex that coincides with a vertex of *A***




(a) Plane $Q$ is defined by an edge of $A$ and edge $e$ of $B$.

(b) $Q$ is intersected with the faces and edges of $A$ incident to $v$.

(c) The closest sector boundary is in a face of $A$ and $fd$ points below the face.

(d) Closest sector boundary is an edge of $A$ and $fd$ splits a volume-enclosing face-pair.

**Figure 48. Transferring a vertex that coincides with a vertex of *A***

A bound on the time required by Algorithm 16 is obtained by observing that the algorithm examines every directed edge of $B$ incident to a vertex $v$ of $A \cap^* B$ once in each of Steps (4), (6), (7), and (8). For each edge of $B$ incident to $v$, the algorithm examines every directed edge adjacent to the coincident vertex of $A$ once in each of Steps (2), (3), and (4). The total time required by Algorithm 16 to transfer edges of $A \bigcap^* B$ incident to vertices of $B$ is

$$O\left( \sum_{v}^{Vertices(B)} D_v + E_v D_A \right) = O(D_A D_B);$$

and the total time required by Algorithm 16 to transfer edges of $A \bigcap^* B$ adjacent to vertices of $A \bigcap^* B$ that are contained by edges of $B$ is

$$O\left( \sum_{e}^{Edges(B)} D_e + D_A \right) = O(D_B + D_A E_B).$$

After all relevant edges and vertices have been transferred to $B$, the boundaries of the faces of $A \bigcap^* B$ that are either in $A$ and on the surface of $B$ are traced out using the techniques of the previous section. The resulting faces of $A \bigcap^* B$ are bounded by cycles consisting of transferred edges and vertices, as well as edges and vertices of $B$ entirely inside $A$. In order to construct these cycles, transferred vertices that coincide with points in the interiors of edges of $B$ must be sorted along the edges of $B$. Every edge of $B$ might intersect $O(E_A)$ edges and faces of $A$ (recall the analysis of Algorithm 10 on page 58), and so it requires time $O(E_B E_A \log E_A)$ to sort vertices of $A \bigcap^* B$ along the edges of $B$. Once these points have been sorted, the cycles of $A \bigcap^* B$ in faces of $B$ that intersect the surface of $A$ are constructed by traversing between edges of $B$ and transferred edges of $A \bigcap^* B$. As described in the previous section, this traversal requires time $O(E_A D_B)$.

After the edges and vertices of $A \bigcap^* B$ have been transferred to faces of $B$ and the traversal of the bounding cycles has been performed, it is possible that there are cycles of the respective faces of $B$ that lie in the interior of $A$, and hence have yet not been added to $A \bigcap^* B$. In order to determine if this is the case, a cycle containment test like Algorithm 12 on page 63 is done. This containment test could work in one of two ways:

(1) A cycle of a face $g$ of $B$ can be tested for containment by $A$ by testing to see if the cycle is contained in the interior of the faces of $A \bigcap^* B$ bounded by cycles and isolated vertices of $A \bigcap^* B$ contained in $g$. Testing containment in this way also tests for containment in $A$, but the test is very expensive: There might be $O(E_A[E_g + V_g])$ directed edges and vertices of $A \bigcap^* B$ contained in $g$ and each of the $O(E_g + V_g)$ containment tests would therefore require time

$O(E_A[E_g + V_g])$. Hence it would require time $O(E_A D_B^2)$ to test for the containment of all directed edge cycles and isolated vertices of $B$ by $A$.

(2) Alternatively, each containment test can implemented by intersecting the plane of $g$ with solid $A$ and then testing to see if each cycle or isolated vertex of $g$ is contained in the **in**$A$ subsets of the constructed cross-section. Construction of the cross-section of solid $A$ requires time $O(D_A \log D_A)$, and each containment test additionally requires time $O(E_A)$. Using this method to test all of the cycles and isolated vertices of $B$ for containment by $A$ requires time

$$O\left( \sum_{g}^{Faces(B)} D_A \log D_A + E_A[E_g + V_g] \right) = O(F_B D_A \log D_A + E_A D_B).$$

Note that containment is tested in the actual implementation of the intersection algorithm using the faces of $A \cap^* B$ and not the cross-sections of $A$.

Finally, all faces of $B$ that lie completely in the interior of A and are adjacent to edges of $A \cap^* B$ are also faces of $A \cap^* B$. These faces are discovered by breadth-first search (as in the previous section) and are added to $A \cap^* B$. This transitive closure requires time $O(|B|)$. Note that this closure operation can obviate some of the containment tests described above.

Summarizing this section, Steps (4) and (5) of Algorithm 5 on page 49 require time $O(D_B D_A \log D_A)$ to find the boundaries of faces of $A \cap^* B$ that are in the interior of $A$ and on the surface of $B$.

## *4.6 Constructing the Star-Edge Representation of $A \cap^* B$*

At this point in the algorithm, all of the bounding directed-edge cycles and isolated vertices of $A \cap^* B$ that result from intersecting shells of $A$ and $B$ have been constructed. These cycles and isolated vertices belong to four types of faces of $A \cap^* B$ :

(i) faces constructed by subdividing a face of $A$ that intersects the surface of $B$;
(ii) faces constructed by subdividing a face of $B$ that intersects the surface of $A$;
(iii) faces of $A$ that are in the interior of $B$; and
(iv) faces of $B$ that are in the interior of $A$.

Cycles and isolated vertices of faces of Types (i) and (ii) might not all belong to the same Star-Edge face of $A \cap^* B$ (see Figure 49 on page 77) and so it is necessary to find the cycles and isolated vertices that belong to connected subsets of the subdivided face. This can be done using either a union-find program [Aho, Hopcroft, and Ullman, 1983] (see Algorithm 17 on page 77) or point location [Kirkpatrick, 1983] (see Algorithm 18 on page 79). (Algorithm 17 was the one actually implemented, but both versions are presented here.)

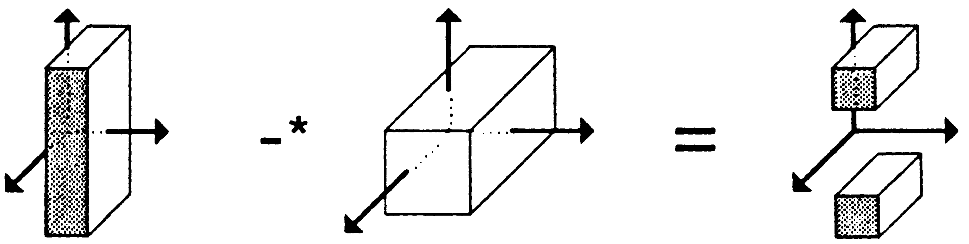


**Figure 49. A subdivided face may not be a connected region**

Given a collection **C** of cycles and isolated vertices in a subdivided face of $A$ or $B$, the cycles and isolated vertices contained in the connected subsets of the subdivided face are found using the following algorithm.

First, form the collection **F** of *tentative faces* by connecting the cycles of **C** into components. Each isolated vertex is also a tentative face. Repeat the following procedure until every tentative face of **F** has been marked as examined.

(1) Select two unexamined tentative faces from **F** (or one examined and one unexamined tentative face if this is not possible), choose a point on each of them, and intersect a line containing the two points with all tentative faces of **F**.

(2) Sort the intersection points along the line and subdivide the line into segments.

(3) For each line segment in the interior of two different tentative faces, unify the two tentative faces and mark them as examined. The local analysis performed is identical to that of Algorithm 6 on page 52.

The unified tentative faces are Star-Edge faces of $A \cap^* B$.

**Algorithm 17. Finding connected subsets of a subdivided face using a union-find program**

If there are $n$ cycles and isolated vertices in the collection **C** of Algorithm 17, then **C** will be intersected with at most $\frac{n+1}{2}$ lines. If there are $D$ directed edges and vertices in **C**, then Step (1) requires time $O(D)$ to intersect a single line with **C**, and Step (2) requires time $O(D \log D)$ to sort the intersection points along a single line. Recalling the analysis of Algorithm 6 on page 52, we see that for each intersected line, Step (3) requires time $O(D)$ to do all of the neighbourhood analyses along a single line. Finally, Step (3) does a total of $O(nD)$ union and find operations, which require a total time of $O(nD\alpha[nD])$. time.[18]

In total, Algorithm 17 requires time

$$O(nD \log D + nD\alpha[nD]) \quad = \quad O(nD \log D)$$

to find the connected subsets of a subdivided face. Each subdivided face $f$ of $A$ has $n = D = O(E_B[E_f + V_f])$, and ach subdivided face $g$ of $B$ has $n = D = O(E_A[E_g + V_g])$. In order to find the connected subsets of all subdivided faces of $A$ and $B$, Algorithm 17 requires time

$$O(D_A{}^2 D_B{}^2 \log[D_A D_B]).$$

If the problem solved in Algorithm 17, finding the connected subsets of the subdivided faces of $A$ and $B$, is rephrased as a point location problem, there is an algorithm that solves the problem in time $O(D_A D_B \log[D_A D_B])$. The idea is as follows: The cycles and isolated vertices of a face are either "outer" or "inner" cycles of the face (see Figure 50) and the connected subsets of a subdivided face can be found by apportioning inner cycles of the subdivided face to outer cycles of the face (see Algorithm 18 on page 79).

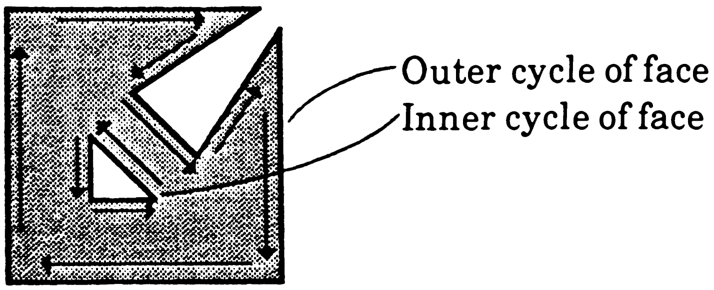



**Figure 50. Outer and Inner cycles of a face**

---

[18] Function $\alpha(n)$ grows so slowly that for feasible $n$, $\alpha(n)$ is constant.

Given a collection **C** of cycles and isolated vertices in a subdivided face of $A$ or $B$, the cycles and isolated vertices contained in the connected subsets of the subdivided face are found using the following algorithm.

First, form the collection **F** of *tentative faces* by connecting the cycles of **C** into components. Each isolated vertex is also a tentative face. Then perform the following procedure:

(1) Find an extreme point of each tentative face and hence determine if the tentative face contains an outer cycle (by a local analysis as in Algorithm 6 on page 52). In this way divide the tentative faces into outer components and inner components.

(2) Triangulate the planar subdivision comprising the outer components.

(3) Pick a point (say a vertex $v$) from each inner component, and search the triangulated subdivision for the outer component that bounds a region containing $v$.

Each outer component together with its contained inner components, forms the boundary of a Star-Edge face of $A \cap^{*} B$.

**Algorithm 18. Finding connected subsets of a subdivided face using point-location**

If a subdivided face has $n$ directed-edges and vertices, and $D$ directed edge cycles and isolated vertices, then Step (1) of Algorithm 18 requires $O(D)$ time. By using an algorithm of Preparata and Shamos [1985] to triangulate a planar subdivision with $m$ vertices in time $O(m \log m)$, Step (2) requires time $O(D \log D)$. Finally, using an algorithm of Kirkpatrick [1983] that locates a point in a triangulated subdivision with $m$ vertices in time $O(\log m)$ after $O(m \log m)$ preprocessing, Step (3) requires time $O(D \log D + n \log D)$ to apportion inner cycles of the subdivided face to outer cycles of the face.

Each subdivided face $f$ of $A$ has $n = D = O(E_B[E_f + V_f])$, and each subdivided face $g$ of $B$ has $n = D = O(E_A[E_g + V_g])$, and so for all subdivided faces of $A$ and $B$, Algorithm 18, which is dominated by Step(3), requires time

$$\begin{aligned} &O(E_B D_A \log[E_B D_A] + E_A D_B \log[E_A D_B]) \\ = \; &O(D_A D_B \log[D_A D_B]). \end{aligned}$$

## 4.7 *Intersecting Solids with Multiple Shells*

The presentation of the regularized intersection algorithm thusfar concentrated on intersecting the shells of a solid, *i.e.*, if the shells of $A$ and $B$ intersect, Steps (1-5) of Algorithm 5 on page 49 assemble pieces of the intersecting shells into

Star-Edge representations of the shells of $A \cap^* B$. In order to intersect multiple-shelled solids $A$ and $B$, two problems must be solved:

(1) It is possible that there are shells of $A$ that do not intersect the surface of $B$ and shells of $B$ that do not intersect the surface of $A$, yet the non-intersecting shells are contained in $A \cap^* B$ (Step [6] of Algorithm 5). An algorithm that determines which of these non-intersecting shells form part of the boundary of $A \cap^* B$ does the three dimensional analogue of the cycle-containment test of Algorithm 12 on page 63.

(2) It might be necessary to determine which shells of $A \cap^* B$ bound each connected component of $A \cap^* B$. (This is the three-dimensional analogue to finding the connected subsets of the subdivided faces of $A$ and $B$. described in Algorithm 17 on page 77 and Algorithm 18 on page 79).

Solutions to these two problems are now described.

Problem (1) is solved by first identifying candidate shells of each solid that may be contained in the other solid and then performing *shell-containment tests*. The algorithms of "Constructing the Star-Edge Representation of $A \cap^* B$" on page 76 can be easily modified to detect shells of $A$ and $B$ that might be shells of $A \cap^* B$. Specifically, Algorithm 17 on page 77 (or Algorithm 18 on page 79) is modified to identify as candidates those shells of solid $A$ (resp. $B$) that did not contribute any boundary points to faces constructed using the algorithms of "Intersecting Faces of Solid $A$ with Cross-Sections of Solid $B$" on page 57 (resp. "Intersecting Faces of Solid $B$ with Solid $A$" on page 65). Once a set of candidate shells has been identified, these shells are checked for containment using an algorithm analogous to the cycle-containment test of Algorithm 12 on page 63, *i.e.*, a shell of $A$ is in the interior of $B$ if a point on $A$ is in the interior of $B$ (and vice versa). This shell-containment test is described in Algorithm 19 on page 81.

To test for containment of a shell $s$ by solid $S$, perform the following steps:

(1) Pick a point $u$ on $s$ and a point $v$ on one of the shells of $S$.

(2) Construct a plane $Q$ containing $u$ and $v$; intersect $S$ with $Q$ (constructing the cross-section $G_Q$ of $S$ with $Q$; see "Computing Cross-Sections of Solid $B$" on page 51).

(3) Use the cycle-containment test (Algorithm 12 on page 63) to determine if point $u$ is inside a subset of $G_Q$ that is in the interior of $S$.

**Algorithm 19. Shell Containment Test**

If solid $S$ has $D_S$ directed edges, then Step (2) of Algorithm 19 constructs the cross-section $G_Q$ of $S$ in time $O(D_S \log D_S)$, and Step (2) tests for the containment of point $u$ in an **in**$S$ subset of $G_Q$ in time $O(E_S)$. The number of shells of a solid is bounded by the number of faces of that solid, and so Algorithm 19 can test for containment of all shells of $A$ by $B$ and all shells of $B$ by $A$ in time

$$\begin{aligned} & O(F_A D_B \log D_B + F_B D_A \log D_A) \\ = \; & O(D_A D_B \log[D_A D_B]). \end{aligned}$$

Problem (2), finding the connected components of $A \cap^* B$, is solved by generalizing the union-find program of Algorithm 17 on page 77. Recall that Algorithm 17 finds the connected subsets of the subdivided faces of $A$ and $B$ in time $O(D_A^2 D_B^2 \log[D_A D_B])$.. Unfortunately, Algorithm 18 which solves the same problem in time $O(D_A D_B \log[D_A D_B])$ using the triangulation-refinement technique of [Kirkpatrick, 1983] does not seem to generalize to three dimensions. In contrast, just as Algorithm 19 generalizes the cycle-containment test of Algorithm 12 on page 63, Algorithm 17 generalizes to find the connected subsets of $A \cap^* B$ (see Algorithm 20 on page 82).

Given a Star-Edge representation of solid $A \bigcap^* B$, the shells of each connected component of $A \bigcap^* B$ are found using the following algorithm.

First, assume that each shell of $A \bigcap^* B$ bounds a different *tentative component*, and denote the collection of tentative components by $S$. Repeat the following procedure until every component of $S$ has been marked as examined.

(1) Select two unexamined shells from $S$ (or one examined and one unexamined shell if this is not possible) and choose a point on each of them. Construct an oriented plane $Q$ containing the two points, and construct the cross-section $G_Q$ of $A \bigcap^* B$ (see "Computing Cross-Sections of Solid $B$" on page 51). Intersect the line containing the two points with the directed edges and vertices of $G_Q$.

(2) Sort the intersection points along the line and subdivide the line into segments.

(3) For each segment in the interior of two different tentative components, unify the two tentative components and mark the constituent shells as examined. The local analysis performed is identical to that of Algorithm 6 on page 52.

The resulting tentative components are the connected components of $A \bigcap^* B$.

**Algorithm 20. Finding the connected components of $A \bigcap^* B$**

The analysis of asymptotic running time of Algorithm 20 is similar to that of Algorithm 17. If there are $n$ shells in the Star-Edge representation of $A \bigcap^* B$ then $A \bigcap^* B$ is intersected with at most $\frac{n+1}{2}$ oriented planes. If there are $D$ directed edges and vertices in $A \bigcap^* B$, Step (1) requires time $O(D \log D)$ to construct a single cross-section $G_Q$ and then time $O(D)$ to intersect $G_Q$ with a line. Step (2) requires time $O(D \log D)$ to sort the intersection points along a single line. Step (3) requires time $O(D)$ to do all of the neighbourhood analyses along a single line. Finally, Step (3) does a total of $O(nD)$ union and find operations, which require a total time of $O(nD\alpha[nD])$.

In total, Algorithm 20 requires time

$$O(nD \log D + nD\alpha[nD]) \quad = \quad O(nD \log D)$$

to find the shells of the connected components of $A \bigcap^* B$. For $A \bigcap^* B$, $D = O(D_A D_B)$. Usually $n$ is small, and then Algorithm 20 requires time $O(D_A D_B \log[D_A D_B])$, although $n$ can be as large as $O(F_A F_B)$.

If $A$ and $B$ are single shelled solids then Algorithm 20 is unnecessary. If we know which of $A$ and $B$ is bounded, then there is a simple algorithm, given the shells of $A \bigcap^* B$, to determine which shells bound each of the connected components of $A \bigcap^* B$ (see Algorithm 21 on page 83).

Given Star-Edge representations of single-shelled solids $A$ and $B$, and given the Star-Edge representations of the shells of $A \bigcap^* B$, the shells of the connected components of $A \bigcap^* B$ can be found using the following procedure:

First, by local analysis at extreme vertices, determine if $A$ and $B$ are bounded. Then apportion the shells to components according to the following cases:

(i) if $A$ and $B$ are both bounded solids, then $A \bigcap^* B$ is a collection of bounded, single-shelled components;

(ii) if neither $A$ nor $B$ is bounded, then $A \bigcap^* B$ is a collection of single-shelled components, one of which is unbounded; and finally

(iii) if $A$ is bounded and $B$ is not, then $A \bigcap^* B$ is either a collection of bounded, single shelled components or a 2-shelled solid bounded by the shells of $A$ and $B$.

**Algorithm 21. Intersecting two single-shelled solids**

In order to analyze the asymptotic complexity Algorithm 21, we first see how to determine whether a shell bounds a bounded volume. We determine if a shell $S$ bounds a bounded volume by finding an extreme vertex $v$ of $S$ in an arbitrary direction $d$, and constructing an oriented plane $Q$ with normal $d$ that contains $v$. Then we use Algorithm 8 on page 56 to determine if $v$ is **in**$S$, **on**$S$, or **out**$S$. Finally, shell $S$ bounds an unbounded volume if $v$ is **in**$S$ or if $v$ is **out**$S$ and has a face that lies in $Q$. It requires time $O(V_S)$ to find vertex $v$, and Algorithm 8 additionally requires $O(D_v)$ time; finally, if $v$ is **out**$S$, it requires time $O(F_v)$ to determine if there is a face of $v$ oriented oppositely to $Q$.

**Lemma 8.** If $A$ and $B$ are single-shelled solids, then Algorithm 21 finds the shells of the connected components of $A \bigcap^* B$ in time $O(D_A D_B)$.

**Proof.** Case (i) is trivial because $A \bigcap^* B$ is bounded. Case (ii) follows by examining the regularized union of $\neg^* A$ and $\neg^* B$, which is bounded, but may have holes. Case (iii), which computes $A -^* B$, yields a 2-shelled solid if the shells of $A$ and $B$ do not intersect, and a collection of bounded, single-shells components otherwise.

It requires time $O(D_A + D_B)$ to determine if $A$ and $B$ are bounded, and it requires $O(D_A D_B)$ to find out which component of $A \bigcap^* B$ is unbounded. ■

Thus, given Star-Edge representations of single-shelled solids $A$ and $B$, a Star-Edge representation of the components of $A \bigcap^* B$ can be constructed in time $O(D_A D_B \log[D_A D_B])$.

## 4.8 Summary

The main theorem of this chapter, that Star-Edge representations of the shells of $A \bigcap^* B$ can be constructed in time $O(D_A D_B \log[D_A D_B])$ from Star-Edge representations of the shells of $A$ and $B$, has been proven. This bound might not optimal, because the only lower bound known for intersecting $A$ and $B$, given their Star-Edge representations, is $\Omega(D_A D_B)$. The problem of apportioning the shells of $A \bigcap^* B$ to the components of $A \bigcap^* B$ is more difficult. We saw that if there are $n$ shells in $A \bigcap^* B$, then the problem of constructing Star-Edge representations of the components of $A \bigcap^* B$ can be solved in time $O(nD_A D_B \log[D_A D_B])$ using a union-find algorithm. We also saw that if $A$ and $B$ are single-shelled solids, then we can construct Star-Edge representations of the components of $A \bigcap^* B$ in time $O(D_A D_B \log[D_A D_B])$.

# Chapter 5. Incidence Tests

In order to compute the intersection of two solids, it is necessary to identify which points on the surfaces of these solids coincide. Incidence tests work by comparing features (vertices, edges, and faces) of representations of the two solids. Implementations of these tests often only approximate true incidence. If numerical data such as vertex coordinates are represented using finite precision, or the tests themselves are implemented using finite-precision arithmetic, we can never ask if two features are truly incident. We can only ask if they are sufficiently close that we might presume incidence.

This chapter then, contains a description of new finite-precision incidence tests used by an implementation of the new intersection algorithm of "Chapter 4. Intersecting Solids". (Experiments which evaluate the utility of the incidence tests are given in Appendix D.)

## *5.1 Incidence in a Finite-Precision World*

Before the new incidence tests can be presented, it is necessary to show why testing for incidence is difficult in a finite-precision world. Suppose that we are given the equation of a plane $P$, and we are given a boundary representation $A$ of a solid, and we would like to say that a feature of $A$ is "deemed incident" with $P$ if the feature is very close to $P$. Using proximity to define incidence is necessary if $P$ or the features of $A$ are given using finite precision, or if we cannot implement a test for exact incidence. Assume then, that edge $e$ of $A$ is defined by the coordinates of its endpoints $v$ and $w$, and $w$ is far from $P$ but $v$ is sufficiently close to $P$ that we think $v$ is incident with $P$. Then there are three possibilities:

(1) $e$ intersects $P$ at $v$;

(2) $e$ intersects $P$ at an interior point, and $v$ and $w$ are on opposite sides of $P$; or

(3) $e$ does not intersect $P$, and $v$ and $w$ are on the same side of $P$.

Whichever of (1)-(3) we choose to be "true" is unimportant. Because we are defining incidence using proximity, $e \cap P$ cannot be computed independently of the incidence of $v$ with $P$. In general, the result of an incidence test must be *consistent* with the results of other incidence tests. Note that the sequence in which the incidence tests are done is also important. For example, deciding that $e$ intersected $P$ at an interior point is inconsistent with the decision that $v$ and $w$ are on the same side of $P$. Such consistency is guaranteed if we can test for exact incidence, but not if we define incidence using proximity. If we are to implement an algorithm that intersects two solids, and the geometric data in the input boundary representations are approximated using finite precision, then testing for incidence is difficult.

**Definition.** If the embeddings in $\mathbf{R}^3$ of the features of solids $A$ and $B$ are approximated using finite precision, then features $a$ and $b$ of solids $A$ and $B$, respectively, are *proximal* to a tolerance $\varepsilon$ if the distance $d(a,b)$ is at most $\varepsilon$. Features $a$ and $b$ are *deemed incident* if they are proximal and their deemed incidence does not logically contradict the non-proximality of any other features of $A$ and $B$.

For brevity, when we say that two features are *incident* we will mean that to say the features are deemed incident.

The above definition suggests using a strategy of symbolic inference to test if two features are incident, summarized as Algorithm 22.

(1) **Test for proximity.** If features $a$ and $b$ of B-Reps $A$ and $B$, respectively, are separated by more than $\varepsilon$ then $a$ and $b$ are not incident.

(2) **Search for contradiction.** Otherwise, if $A$ and $B$ have features $a'$ and $b'$, respectively, that are separated by more than $\varepsilon$, but whose incidence is logically implied by the incidence of $a$ and $b$, then $a$ and $b$ are not incident.

(3) **Conclude incidence.** Finally, if no such features can be found, then $a$ and $b$ are incident.

**Algorithm 22. Testing for deemed incidence**

Once we have decided that $a$ and $b$ are not incident, we would like to determine their positions in some local coordinate system. If $a$ and $b$ are not incident because they are separated by more than $\varepsilon$, then this is straightforward. For example, if Algorithm 22 reported that a vertex is far from an oriented plane, we can easily determine if the vertex is above or below the plane. In contrast, if Algorithm 22 reported that features $a$ and $b$ are not incident because of a contradiction, then the relative locations of $a$ and $b$ are found by working backwards along the chain of reasoning that led to the contradiction. We will see when we discuss the actual incidence tests that this backwards reasoning can get quite detailed.

We will be using the incidence tests described in this chapter in an implementation of a regularized intersection algorithm for solids, and it is possible that an incidence test can make a "mistake" and cause the intersection algorithm to fail. Suppose that an incidence test is searching for contradictory evidence to disprove the incidence of two features, and in this search discovers that regardless of the outcome of the test, an inconsistent incidence decision will have been made, *i.e.* two deemed incidences are contradictory. For example, a consequence of one incidence test might be that a vertex is above a plane, and a consequence of another incidence test might be that the vertex is below the plane. In the context of this thesis, we say that if an intersection algorithm is provably correct using a model of exact arithmetic and the algorithm is implemented to use the incidence tests of this chapter then the algorithm succeeds if no mistakes are made.

Because our finite-precision tests do not report true incidence, we cannot use them in an algorithm and expect them to function like true incidence tests. For example, if we use the incidence tests of this chapter in an intersection algorithm for solids, we might decide that $A \bigcap B = A$, and that $B \bigcap A = B$. Both answers may be correct because every feature of $A$ might be within $\varepsilon$ of the corresponding feature of $B$. This example does illustrate, that unlike true incidence, deemed incidence is not symmetric. For example, just because we can deem vertex $v$ incident with vertex $u$, we cannot deem $u$ incident with $v$. More importantly, deemed incidence tests are based on proximity, which is not a transitive relation, and so unlike true incidence, deemed incidence is not transitive.

## 5.2 *Simplifications and Assumptions*

The idea that symbolic inference can make incidence tests more reliable is new, and one of the objectives of this thesis is to demonstrate that it is practical. In order to do this, the new incidence tests that use symbolic inference should be either provably correct, or at least more experimentally reliable than incidence tests that do not use symbolic inference. Furthermore, the new tests should be implementable and not too inefficient. For example, one could hypothesize the existence of two different kinds of finite-precision incidence tests. Given Star-Edge representations of solids $A$ and $B$:

(1) A very simple incidence test that does not test for deemed incidence could declare that a feature of $A$ is coincident with a feature of $B$ if the distance between the two features is less than a tolerance $\varepsilon$.

(2) A very complicated incidence test might use the distance of every feature of $A$ from every feature of $B$ in order to determine the deemed incidence of any two of the features.

Certainly, the naive test (1) is efficient, but it will make errors and so may not be reliable. The sophisticated incidence test (2) might be provably correct, totally reliable, but probably inefficient. In order to make incidence testing via symbolic inference practical, we would like to develop incidence tests that are more sophisticated than Test (1) above, but less complicated than Test (2). We are willing to report an occasional mistake if the incidence tests are relatively efficient and easy to implement. In particular this means that the incidence tests we adopt are neither complete nor provably correct. The experiments described in Appendix D demonstrate when the new incidence tests can be expected to be reliable and when they might fail. The experiments also show that the new incidence tests are practical.

In the context of this thesis, no attempt has yet been made to analyze the stability of numerical computations used by the incidence tests. If the magnitude of a numerical computation, *e.g.*, an inner product of two vectors, is less than $\varepsilon$, the computation is assumed to be unreliable and a symbolic inference determines the sign of the unreliable quantity. (The sign of the quantity is sufficient for our purposes if the quantity is nearly zero). It is important to choose $\varepsilon$ small enough so that the result of an incidence test is meaningful. It is also important that features of a solid be larger than $\varepsilon$. We use the idea of a *minimum feature size* of a boundary representation [Segal and Séquin 1985] and require that every edge of a solid have length at least $\tau$, and the angle between two incident faces or edges must be at least $\alpha$ and at most $\pi - \alpha$. These two conditions guarantee that, for example, we can sort edges around a common vertex and that at most one vertex from a solid is in the same $\varepsilon$-ball. For the experiments described in Appendix D, the solids have approximately unit size, finite-precision quantities are represented using 53 bits of precision, and values of $\varepsilon$ as large as $10^{-2}$ and as small as $10^{-6}$ were used with success. Much larger values of $\varepsilon$ violated minimum feature-size assumptions in the input boundary representations, and much smaller values of $\varepsilon$ caused numerical instability.

We saw in "Chapter 2. Representing Solids" that the Star-Edge representation can describe solid boundaries that are complicated non-manifold surfaces. However (using finite precision) it is unlikely that equations of planes can be given with sufficient precision to correctly describe high-valence vertices. For example, four planes meant to intersect at a point probably intersect as two sets of three planes connected by a very tiny line segment. Thus many symbolic inferences are restricted to valence-three vertices because inferences that use high-valence vertices may fail. More generally, we assume that vertices are defined by the intersection of three planes, and edges are defined by the intersection of two planes.[19] In practice, restricting the inferences has not proved to be a great problem. The first test object whose construction is described in Appendix D has vertices with valence 12. The intersection algorithm can fail if two solids coincide near a high-valence vertex, and neither numerical tests nor simple inferences prove or disprove a deemed coincidence.

The remainder of this chapter contains a description of the incidence tests, which compute the incidence of features of solid $B$ with solid $A$. A related problem, sorting, is also discussed; in particular, the intersection points of a line with a solid can be arbitrarily close, and symbolic reasoning may be necessary to order these points along the line.

**Notation.** The named features (vertices, edges, and faces) of solid $B$ will be suffixed with a "′", *e.g.*, $e'$ is an edge of $B$ and $e$ is an edge of solid $A$.

---

[19] The key observation we make is that certain geometric data in a boundary representation are "original," and certain redundant data inferred. In a scheme related to ours, Sugihara [1987] described a technique where plane-equation coefficients are discrete quantities (hence error-free), and redundant data is approximated from the plane equations using finite precision. A major disadvantage to Sugihara's technique is that rotational transformations are difficult to apply to represented solids because rotated planes tend to have transcendental coefficients. Note that our scheme does not have this disadvantage because we always approximate planes using finite precision. In difference to this, Segal and Séquin [1985] discuss how to use least-squares approximation to compute plane equations that fit a collection of vertices. Yamaguchi and Tokieda [1985] and Paoluzzi, Ramella, and Santarelli [1986] avoid the problem of redundancy altogether by using triangular faces.

## 5.3 Vertex-Incidence Tests

Given face $f$ of solid $A$ and vertex $v'$ of solid $B$, we classify $v'$ with $f$: first, we ascertain whether $v'$ is above, below, or on the plane $P$ of $f$; second, if $v'$ is on $P$, we ascertain whether $v'$ is contained in the line $P \cap Q$, where $Q$ is the plane of a face adjacent to $f$; finally, if $v'$ is in $P \cap Q$, we ascertain whether $v'$ is in the interior of the edge $e$ of $f$ contained by $P \cap Q$ or coincident with an endpoint of $e$ (see Figure 51).

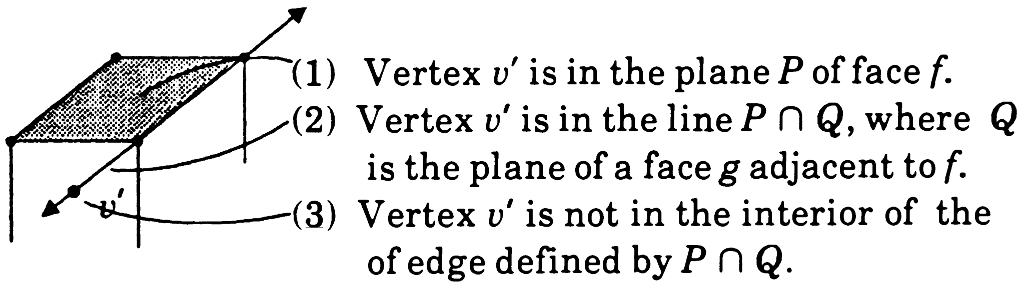
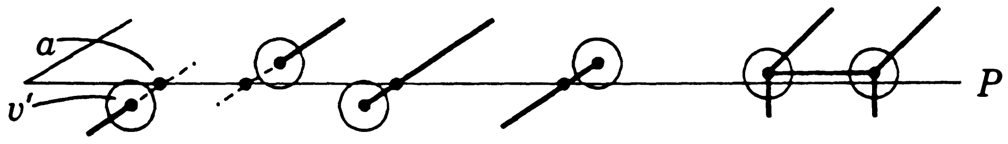



**Figure 51. Organizing vertex-incidence tests**

The first test to be performed determines whether vertex $v'$ is above, below, or on plane $P$. First, $v'$ is not on $P$ if the computed distance between $v'$ and $P$ is large, *i.e.*, greater than $\varepsilon$. If $v'$ is near to $P$, then $v'$ does not lie on $P$ if there is an edge adjacent to $v'$ that does not intersect $P$ within $\varepsilon$ of $v'$. If no such edge exists, then $v'$ and $P$ are deemed incident (see Algorithm 23 on page 91).

Compute the distance between vertex $v'$ and plane $P$ by substituting the coordinates of $v'$ into the implicit equation of $P$. If this distance is larger than $\varepsilon$, then $v'$ is off $P$, and the sign of this distance classifies $v'$ as either above or below $P$. If the computed distance is smaller than $\varepsilon$ and there is an edge adjacent to $v'$ that does not intersect $P$ within $\varepsilon$ of $v'$, then $v'$ is not on $P$:

> Consider edge $e'$ adjacent to $v'$ and vertex $w'$, where $w'$ has been shown to be off $P$ by Algorithm 23 on page 91. Compute the intersection of $e'$ with $P$ by intersecting $P$ with the two planes that define $e'$. If the computed intersection point $a$ is not within $\varepsilon$ of $v'$ then $v'$ is not on $P$; Furthermore the classification of $w'$ is used to determine whether $v'$ is above or below $P$: if $a$ is in the interior of $e'$, then $v'$ and $w'$ are on opposite sides of $P$; otherwise $v'$ and $w'$ are on the same side of $P$.




If the above procedure does not classify $v'$ as above or below $P$, then $v'$ lies on $P$, because all edges adjacent to $v'$ either lie on $P$ or intersect $P$ within $\varepsilon$ of $v'$.

**Algorithm 23. Testing for vertex/plane incidence**

If Algorithm 23 reports that vertex $v'$ lies in the plane $P$ of face $f$, then $v'$ may be incident with an edge or vertex of $f$. This cannot happen if $v'$ does not lie in the plane of any face adjacent to $f$ (Algorithm 24).

Given non-coincident planes $P$ and $Q$ and a vertex $v'$ on $P$, $v'$ is in the line $P \cap Q$ iff the test for vertex/plane incidence classifies $v'$ on $Q$. If $v'$ is not on $Q$, then $v'$ is either above or below $Q$, and hence to the left or right of $P \cap Q$ (on $P$).

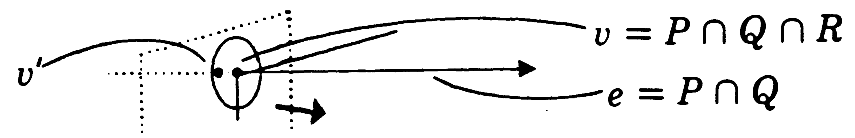
**Algorithm 24. Testing for vertex/line incidence**

If vertex $v'$ is contained in the line $P \cap Q$ that defines the intersection of face $f$ with face $g$, then $v'$ may be in the interior of the edge $e$ adjacent to $f$ and $g$, or coincident with a vertex $v$ of $e$ defined by the intersection of $e$ with a third plane $R$. Algorithm 25 on page 92 tests to see if $v'$ lies in $R$ (using Algorithm 23). If $v'$ lies in $R$ then $v'$ coincides with $v$. Otherwise, $v'$ is in either the exterior of $e$ or the open ray containing $e$ and beginning at $v$. By applying Algorithm 25 to both endpoints of $e$, we determine if $v$ is in the interior of $e$ or coincident with an endpoint.

Given line $P \cap Q$ that defines edge $e$ and contains vertex $v'$, we determine if $v'$ coincides with $v$ or is contained in the open ray beginning at $v$ and containing $e$.

(1) Use the test for vertex/plane incidence (Algorithm 23) to determine if $v'$ is above or below $R$.

(2) If $v'$ is not on $R$, then the classification returned by the test for vertex/plane incidence is used to infer whether $v'$ is in the exterior of $e$ or in the open ray containing $e$ and beginning at $v$. This determination is made by examining the dot product of the normal to $R$ with a vector directed from $v$ to the other endpoint of $e$.




**Algorithm 25. Testing for vertex/ray incidence**

Previously, we observed that our definition of (deemed) incidence is not symmetric. For example, suppose that the test for vertex/plane incidence reports that vertex $v'$ of solid $B$ is incident to each of the defining planes of vertex $v$ of solid $A$, and hence we would deem $v'$ incident with $v$. There is no requirement that $v$ be incident with each of the defining planes of $v'$, and so we might not deem $v$ incident with $v'$. We avoid this possible inconsistency by differentiating between features of $A$ and features of $B$. We ask only if a feature of $B$ is incident with a feature of $A$, and not if a feature of $A$ is incident with a feature of $B$.

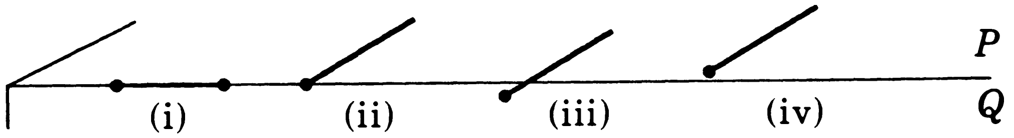
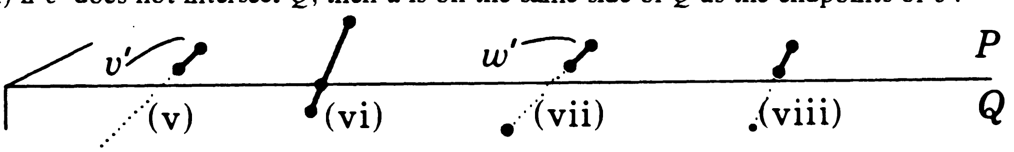
## 5.4 *Edge-Incidence Tests*

In this section we develop the edge-incidence tests used by the intersection algorithm. Given face $f$ of solid $A$ and edge $e'$ of solid $B$, we classify $e'$ with $f$. First, we ascertain whether $e'$ intersects the plane $P$ of $f$; second, if the interior of $e'$ intersects $P$, we ascertain whether $e'$ intersects the line containing an edge of $f$; finally, if $e'$ does intersect this line, we ascertain whether $e'$ intersects a vertex of $f$ or the interior of an edge of $f$. The edge-incidence tests, like the vertex-incidence tests of the previous section, are neither symmetric nor transitive, and so they must be used with care.

We begin by computing the intersection of edge $e'$ with the plane $P$ of a face $f$. This intersection is easily computed using the test for vertex/plane incidence to classify the endpoints of $e'$. A case analysis, given in Algorithm 26, is used to determine how $e'$ and $P$ intersect.

> Intersect edge $e'$ with plane $P$ as follows: First, use the test for vertex/plane incidence to classify both endpoints of $e'$ with $P$; if both endpoints of $e'$ are on $P$, then $e'$ is on $P$; if the endpoints of $e'$ are on opposite sides of $P$, then the intersection of $e'$ with $P$ is computed by intersecting $P$ with the two planes that define $e$; otherwise $e'$ intersects $P$ at a single endpoint or not at all.
>
> **Algorithm 26. Testing for edge/plane incidence**

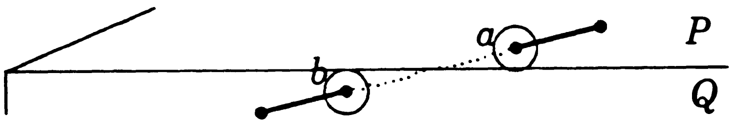
If a single endpoint of edge $e'$ lies in the plane $P$ of face $f$, then the test for vertex/ray incidence is used to finish the incidence computation. Otherwise, $P$ may contain $e'$ or intersect the interior of $e'$. As in the vertex-incidence tests, we ask if $e'$ intersects a line $P \cap Q$, where $Q$ is the plane of a face adjacent to $f$. Algorithm 27 describes the simple cases.

> Given non-coincident planes $P$ and $Q$, and an edge $e'$ that intersects $P$, we compute the intersection of $e'$ with $P \cap Q$ in some special cases. If $e'$ lies on $P$ then:
>
> (i) if $e'$ lies on $Q$, then $e'$ is collinear with $P \cap Q$;
> (ii) if one endpoint $v'$ of $e'$ intersects $Q$, then $e'$ intersects $P \cap Q$ at $v'$;
> (iii) if the interior of $e'$ intersects $Q$ at a point $b$ then $e'$ intersects $P \cap Q$; and
> (iv) if $e'$ does not intersect $Q$, then $e'$ does not intersect $P \cap Q$.
>
> 
> 
>
> Otherwise, the interior of $e'$ intersects $P$ at a point $a$, and:
> (v) if the distance $Q(a)$ between $Q$ and $a$ is large ( $> \varepsilon$) then $e'$ does not intersect $P \cap Q$;
> (vi) if $e'$ is on $Q$, then $e'$ intersects $P \cap Q$;
> (vii) if one endpoint $v'$ of $e'$ intersects $Q$, then $a$ and the other endpoint $w'$ of $e'$ are on the same side of $Q$; and
> (viii) if $e'$ does not intersect $Q$, then $a$ is on the same side of $Q$ as the endpoints of $e'$.
>
> 
> 
>
> **Algorithm 27. Simple cases in testing for edge/line incidence**

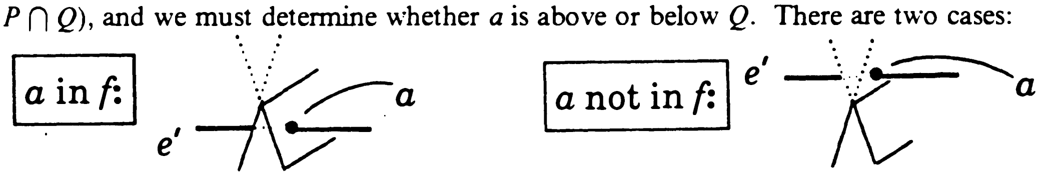
If the cases described in Algorithm 27 do not apply, then the interior of edge $e'$ intersects planes $P$ and $Q$; furthermore, the intersection of $e'$ with $P$ is within $\varepsilon$ of $Q$, and the intersection of $e'$ with $Q$ is within $\varepsilon$ of $P$. If $P$ and $Q$ are nearly coincident it is possible that these two intersection points are not near $P \cap Q$ (Algorithm 28).

If the interior of an edge $e'$ intersects plane $P$ at point $a$ and plane $Q$ at point $b$, then we determine if $e'$ intersects $P \cap Q$:

(1) If $a$ and $b$ are separated by more than $\varepsilon$, then neither $a$ nor $b$ is contained in $P \cap Q$. Furthermore, by ordering $a$, $b$ and the endpoints of $e'$ along $P \cap Q$, we can classify $a$ as above or below $Q$ and $b$ as above or below $P$ (using the test for vertex/plane incidence to classify an endpoint of $e'$ with either $P$ or $Q$).




(2) Otherwise $a$ is within $\varepsilon$ of $b$, and we determine if $a$ and $b$ are near $P \cap Q$ by constructing a plane containing $P \cap Q$ and approximately orthogonal to both $P$ and $Q$ ($S = P + Q$ will do), and computing the distances $S(a)$ and $S(b)$. If both of these distances are less than $\varepsilon$, then $e'$ intersects $P \cap Q$. Otherwise, one of these points, say $a$, is far from $S$ (and $P \cap Q$), and we must determine whether $a$ is above or below $Q$. There are two cases:

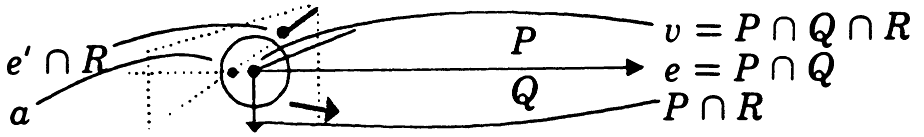



First, we determine if $a$ is in the interior of the face $f$ on $P$: we identify the edge $e$ contained by $P \cap Q$ (adjacent to faces $f$ and $g$) and compute the face direction vector from $e$ into the interior of $f$; next, we compute the dot product of this vector with the normal vector to $S$; and $a$ is in $f$ iff the sign of this dot product agrees with the sign of $S(a)$. Finally, $a$ is below $Q$ iff both or neither of the following is true: (i) the angle subtended by the faces at $e$ is acute; and (ii) $a$ is in the interior of $f$. Once $a$ has been classified as above or below $Q$, $b$ is classified as above or below $P$ from the classifications of the endpoints of $e$ with $P$ and $Q$.

**Algorithm 28. The general case for edge/line incidence**

If edge $e'$ intersects the line $P \cap Q$, then $e'$ may intersect the edge $e$ defined by $P \cap Q$. If a single endpoint $v'$ of $e'$ intersects $P \cap Q$, then the test for vertex/ray incidence (Algorithm 25 on page 92) is used to test if $v'$ intersects the interior of $e$ or an endpoint. Otherwise there are two remaining cases: either the interior of $e'$ intersects $P \cap Q$ at a point $a$ or $e'$ is collinear with $P \cap Q$. Algorithm 29 determines if $e'$ either contains endpoint $v$ of $e$ or intersects the open ray beginning at $v$ and containing $e$. By applying Algorithm 29 to both endpoints of $e$, we determine if $e'$ intersects the interior of $e$ or an endpoint.

Given a line $P \cap Q$ which defines edge $e$ and intersects the interior of edge $e'$ at a point $a$, we determine if $e'$ intersects vertex $v$ of $e$ or the open ray containing $e$ and beginning at $v$. Because $e$ is defined by $P \cap Q$, and $v$ is defined by the intersection of $e$ with a third plane $R$, we compute the incidence of $e'$ with $P \cap R$ and $Q \cap R$. If $e'$ intersects both of these lines, then $e'$ intersects $v$. Otherwise, because $e'$ does not intersect one of these two lines (say $P \cap R$), the edge/line incidence test classified the intersection of $e'$ with $P$ (point $a$) as above or below $R$.




From the classification of point $a$ with $R$ we infer whether $e'$ intersects the open ray containing $e$ and beginning at $v$. This determination is made by by examining the dot product of the normal vector to $R$ with the vector directed from $v$ to the other endpoint of $e$.

**Algorithm 29. Testing for edge/ray incidence**

Algorithm 29 infers that edge $e'$ intersects a vertex $v$ of edge $e$ only if $e'$ intersects all three lines containing edges adjacent to $v$. There is inconsistency if the classifications of $e'$ with these lines do not agree. This inconsistency is reported as a mistake by the implemented incidence tests.

The remaining edge-incidence test arises if the test for edge/line incidence reports that edge $e'$ of solid $A$ is collinear with a line containing an edge $e$ of solid $B$. It is possible that endpoints of $e$ are in the interior of $e'$. This is resolved by the case analysis in Algorithm 30 on page 96.

Given collinear edges $e$ and $e'$, we determine if the endpoints of $e$ intersects $e'$:

(1) Divide the line containing $e$ and $e'$ into five regions corresponding to the left, left-endpoint, interior, right-endpoint, and right regions of $e$.

(2) An endpoint of $e$ is contained by $e'$ if that endpoint is "surrounded" by the endpoints of $e'$ (for example, if one endpoint of $e'$ is in the left region, then we require that the other endpoint is in neither the left nor the left-endpoint regions).

**Algorithm 30. Testing for collinear-edge intersection**

## 5.5 Face-Incidence Tests

In this section the last of the incidence tests, the face-incidence tests, are described. Given *cross-face edge* $\bar{e}$ defined by the intersection of face $f'$ of solid $B$ with plane $P$ of face $f$ of solid $A$ (see Figure 52) we determine if $\bar{e}$ intersects an edge or vertex of $f$. As in the vertex- and edge- incidence tests, $\bar{e}$ cannot intersect an edge or vertex of $f$ if $\bar{e}$ does not intersect a line $P \cap Q$, where $Q$ is the plane of a face adjacent to $f$. We begin by classifying the endpoints of $\bar{e}$ with $P \cap Q$. Because the endpoints of $\bar{e}$ can be either vertices of solid $B$ or points in the interiors of edges of $B$, Algorithm 31 on page 97 uses either the test for vertex/line incidence (Algorithm 24 on page 91) or the test for edge/line incidence (Algorithm 27 on page 93 and Algorithm 28 on page 94) to classify an endpoint of $\bar{e}$ with $P \cap Q$.

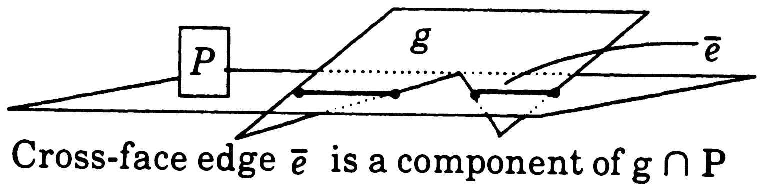
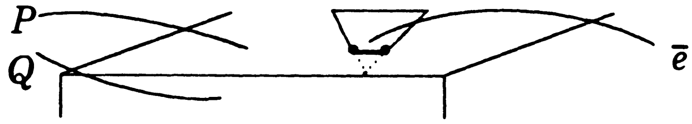


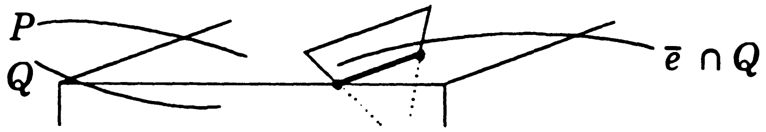

**Figure 52. A cross-face edge**

Given cross-face edge $\bar{e}$ defined by the intersection of face $f'$ (of solid $B$) with the plane $P$ of face $f$ (of solid $A$), and given plane $Q$ not coincident with $P$, we determine if $\bar{e}$ intersects $P \cap Q$. We begin by computing the incidence of the endpoints of $\bar{e}$ with $P \cap Q$ to determine if $\bar{e}$ can intersect $P \cap Q$. There are two cases: if an endpoint of $\bar{e}$ is a vertex, then we use Algorithm 24 on page 91 to test for vertex-line incidence; otherwise, the endpoint of $\bar{e}$ is a point in the interior of an edge $e'$ of solid $B$ and we use Algorithm 27 on page 93 and Algorithm 28 on page 94 to test for edge/line incidence. We identify several cases:

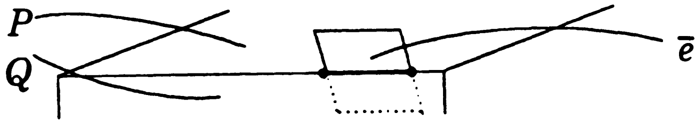
(i) if the endpoints of $\bar{e}$ are on the same side of $P \cap Q$ then $\bar{e}$ does not intersect $P \cap Q$;



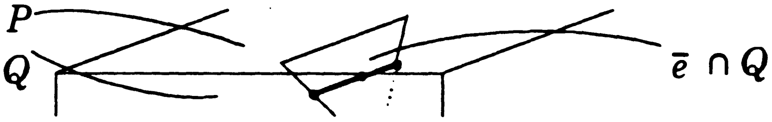

(ii) $\bar{e}$ may intersect $P \cap Q$ at a single endpoint;




(iii) if both endpoints of $\bar{e}$ are in $P \cap Q$, then $\bar{e}$ is contained in $P \cap Q$; otherwise




(iv) the endpoints of $\bar{e}$ are on opposite sides of $P \cap Q$ and so $\bar{e}$ intersects $P \cap Q$ at an interior point.




**Algorithm 31. Testing for cross-face-edge/line incidence**

If Algorithm 31 reports that $\bar{e}$ intersects the line $P \cap Q$. then $\bar{e}$ can intersect the edge $e$ defined by $P \cap Q$: (i) the case where $\bar{e}$ intersects this line at an endpoint is dealt with by using the vertex- and edge-incidence tests already described; (ii) the case where $\bar{e}$ is collinear with $P \cap Q$ is dealt with in Algorithm 32 on page 98, using a case analysis identical to the test for the intersection of two collinear edges of Algorithm 30 on page 96; finally, (iii) the case where the interior of $\bar{e}$ intersects $P \cap Q$ at an interior point (actually a point in the interior of $f'$) is dealt with in Algorithm 33 on page 98.

Given edge $e$ collinear with cross-face edge $\bar{e}$, we determine if the endpoints of $e$ intersect $\bar{e}$:

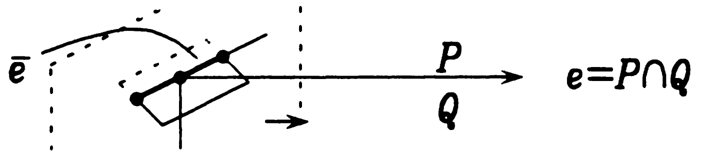
(1) Divide the line containing $e$ and $\bar{e}$ into five regions corresponding to the left, left-endpoint, interior, right-endpoint, and right regions of $e$.

(2) An endpoint of $e$ is contained by $\bar{e}$ if that endpoint is "surrounded" by the endpoints of $\bar{e}$ (for example, if one endpoint of $\bar{e}$ is in the left region, then we require that the other endpoint is in neither the left nor the left-endpoint regions).

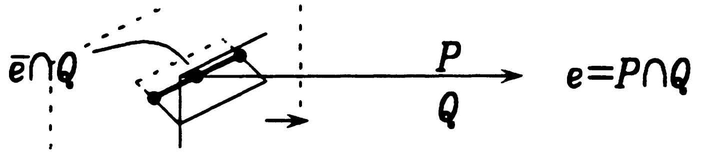
**Algorithm 32. Testing for collinear cross-face-edge/edge intersection**

Given line $P \cap Q$ and cross-face edge $\bar{e}$ defined by the intersection of face $f'$ with $P$, such that the interior of $\bar{e}$ intersects $P \cap Q$ at a point $a$, we determine if $a$ coincides with vertex $v$ of $e$ or is contained in the open ray containing $e$ and beginning at $v$. Because $e$ is defined by $P \cap Q$, and $v$ is defined by the intersection of $e$ with a third plane $R$, we determine whether $\bar{e}$ intersects $v$ by using either *the test for vertex/line incidence* or *the test for edge/line incidence* (on $P \cap R$):

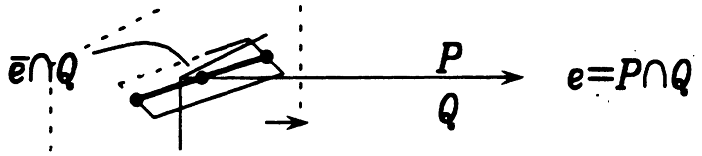
(i) if both endpoints of $\bar{e}$ are contained in $P \cap R$, then $v$ is in the interior of $\bar{e}$;




(ii) if both endpoints of $\bar{e}$ are on the same side of $R$, then we infer whether $\bar{e}$ intersects the interior of $e$;




(iii) if only one endpoint of $\bar{e}$ is on $R$, then we infer whether $\bar{e}$ intersects the interior of $e$; otherwise




(iv) neither endpoint of $\bar{e}$ intersects $R$, and so we use the test for vertex/plane incidence (Algorithm 23 on page 91) to determine whether $v$ intersects the face $f'$ which induced $\bar{e}$. (This is equivalent to determining if the intersections between the plane of $f'$ and the edges adjacent to $v$ are all within $\varepsilon$ of $v$.)

**Algorithm 33. The general case for cross-face-edge/ray incidence**

## 5.6 *Sorting Intersection Points along a Line*

We often wish to sort the intersections of a solid with a line. We can usually perform this sort numerically, *e.g.*, by sorting projections of these points along a principal axis. However, because the new incidence tests can produce intersection points that are not separated by at least $\varepsilon$, we will have to use incidence computations to order such points.

If we use the vertex-, edge-, and face-incidence tests described previously to generate the intersections of a valence-three solid with a line, then the number of different apparent coincidences is quite small. The minimum-feature-size assumption guarantees that two intersection points in the same $\varepsilon$-ball are intersections of the line with features incident to the same edge or vertex of the solid. Two intersection points can be within $\varepsilon$ of each other if they are on: (i) two coplanar edges adjacent to the same vertex; (ii) two faces adjacent to the same edge; or (iii) a face and an edge adjacent to the same vertex. If we abstract the intersection line to correspond to the edge $e'$ of solid $B$ described in the incidence tests, then all three cases are handled by the test for edge/line incidence (Algorithm 27 on page 93 and Algorithm 28 on page 94) and the test for edge/ray incidence (Algorithm 29 on page 95).

# Chapter 6. Conclusions

This thesis has studied the representational requirements, design, and implementation of algorithms for regularized set operations. We began by defining a certain subclass of point sets in $\mathbf{R}^3$, called solids, for studying set operations. After examining issues that arise in representing the boundaries of solids, we described a new boundary representation called the Star-Edge representation. We saw how to construct a Star-Edge representation of the regularized complement of a solid from a Star-Edge representation of the solid. We also examined the problem of determining if two solids represented by their boundaries are identical, and gave an algorithm that solved this problem using the Star-Edge representation. We described a correct algorithm that uses the Star-Edge representation to compute the regularized intersection of two solids. We then discussed how to use symbolic inference in order to implement finite-precision incidence tests more reliably. Finally, we saw experimentally that finite-precision incidence tests which use symbolic inference can be used by the intersection algorithm in order to limit errors due to finite-precision approximations.

The major contributions of this thesis are the new regularized intersection algorithm and the new incidence tests. The asymptotic complexity of the new intersection algorithm shows that the algorithm is efficient, and the experiments described in Appendix D show that the algorithm can be implemented to limit the errors due to finite-precision arithmetic. Finally, the simple preprocessor described in Appendix E eliminates eliminates extraneous computation, and so reduces the actual computation time for many intersections.

After studying the design as well as the implementation of a complicated algorithm, it is useful to re-examine some of the themes adopted as dogma. Not surprisingly, some of of these themes conflict and it is difficult to unify all of them.

## 6.1 Implementing the Intersection Algorithm

Most of the special cases handled by the intersection algorithm are more complicated than were addressed by the inference schemes of the finite-precision incidence tests. (Recall that we limited most inferences used by the new incidence tests to valence-three vertices.) Thus we have described an algorithm that we do not as yet know how to implement reliably using finite-precision arithmetic.

The experimentation described in Appendix D demonstrates that even though the incidence tests are neither complete nor correct, they are empirically reliable. Before it is possible to prove the tests either complete or correct, a suitable notion of correctness is necessary. First attempts at defining correctness of finite-precision implementations of geometric algorithms are given in [Hoffmann, Hopcroft, and Karasick 1988] and [Milenkovic, 1988], by recasting the idea of numerical stability in the context of geometric algorithms.

## 6.2 The Star-Edge Representation

A small set of constraints that describe relationships between the numbers of vertices, faces, edges, and directed edges of a Star-Edge representation would be useful to verify that a boundary representation produced by the intersection algorithm is valid. It has not proved possible to generalize a formulation like Euler's to describe the relationships between the features of a Star-Edge representation. It is unlikely that an exact formula exists because face boundaries are not required to have simple forms like disjoint polygons. For example, we saw a construction of a solid in "Chapter 3. Some Representational Results" of this thesis with O($n$) faces, and O($n^3$) vertices.

The Star-Edge representation is unusual in that it allows a very general notion of face boundary. For example, vertices are not required to be incident to edges. Such "isolated vertices" also detract from the elegance of the intersection algorithm. Moreover, sometimes algorithms that use boundary representations are more efficiently formulated if face-boundaries are simple polygons. Many efficient geometric algorithms cannot be used by the intersection algorithm because Star-Edge face-boundaries do not have simple forms.

## 6.3 Asymptotic Complexity of the Intersection Algorithm

The asymptotic analysis of complexity given in "Chapter 4. Intersecting Solids" established the asymptotic complexity of the implemented intersection algorithm. Certainly a great deal of work could be done to reduce the asymptotic complexity of many of the procedures used by the intersection algorithm. For example, one of these procedures intersects two embedded planar graphs that have a total of $n$ vertices in time $O(n^2 \log n)$. In fact, there is an $O(n^2)$ algorithm that solves the same problem [Chazelle and Edelsbrunner, 1988]. Using this algorithm, it might be possible to implement most of the intersection algorithm in optimal time $\Theta(D_A D_B)$, where $D_A$ and $D_B$ are the numbers of directed edges in the Star-Edge representations of the solids to be intersected.[20]

In most cases the asymptotic analysis of the procedures used by the intersection algorithm A more careful analysis might show that the intersection algorithm is actually faster than the asymptotic complexity proved in "Chapter 4. Intersecting Solids". There are certainly trade-offs in complexity between different phases of the algorithm that were not accounted for in the analysis.[21]

## 6.4 Future Work

The most exciting work that might arise from this thesis is a provably correct implementation of finite-precision incidence tests that use the Star-Edge representation. The experimentation presented in Appendix D is sufficiently promis-

---

[20] Fundamental changes may have to be made to the intersection algorithm in order to achieve an asymptotic running time of $\Theta(D_A D_B)$. For example, recall from "Chapter 4. Intersecting Solids" that the asymptotic complexity of a procedure that intersects a single oriented plane of solid $A$ with solid $B$ is $O(D_B \log D_B)$. The "log" term is a consequence of sorting intersections of the plane with each intersected face of $B$: If an intersected face $g$ of $B$ has $E_g + V_g$ edges and vertices, then $O(E_g + V_g)$ points are sorted. Each of the individual planes of solid $A$ is intersected with solid $A$; and if solid $A$ has $F_A$ faces, then the intersection algorithm requires $O(F_A D_B \log D_B)$ to compute these intersections. In order to reduce the asymptotic complexity of the intersection algorithm to $\Theta(D_A D_B)$, the $F_A$ cross-sections must be computed more efficiently.

[21] For example, there are trade-offs between the time required to subdivide intersected faces of $A$ and $B$ into connected regions and the time required to determine if a point on the boundary of one solid is contained in the interior of a face of the other solid because both procedures examine subdivided faces of $A$ and $B$.

ing that it might be possible to develop a useful definition of correctness for finite-precision geometric algorithms.

The symbolic inference paradigm has only been investigated (in the context of this thesis) to limit the errors due to finite-precision arithmetic. If the regularized intersection algorithm of this thesis were implemented to use rational arithmetic, correctness would be assured, but the algorithm would be much slower because rational calculation is expensive. (For example, if we wish to compute the intersection of $n$ planes, each plane has $k$-bit coefficients, and two $ak$ bit quantities can be multiplied using $a^2$ $k$-bit multiplications, then it requires $O(4^n)$ $k$-bit multiplications to diagonalize the linear system.) Symbolic inference has been used in this thesis to attain robustness, but it should be possible to use symbolic inference in order to empirically lessen the running time of geometric algorithms that use exact, rational arithmetic. (This idea was suggested by Dr. Lee Nackman of IBM Research.)

On a less theoretical level, work is proceeding to generalize some of the incidence tests to work on point sets with curved, rather than polyhedral boundaries. The Star-Edge representation is easily generalized to represent regular sets bounded by oriented quadric surfaces in addition to oriented planes. Given numerical procedures that intersect curved surfaces (see for example, Thomas [1984] or Sinha [1986]), an intersection algorithm can be generalized to intersect objects with boundaries defined using curved surfaces. The challenge is efficient and reliable implementation.

# Appendix A. Some Elementary Topology

A *Topology* for a set $X$ is a family $T$ of subsets of $X$ satisfying the following three properties:

(1) The set $X$ and the empty set $\phi$ are in $T$.

(2) The union of any family of members of $T$ is in $T$.

(3) The intersection of any finite family of members of $T$ is in $T$.

A member of $T$ is called an *Open Set*. An open set that contains a point $p$ is called a *Neighbourhood* of $p$, denoted $Nbhd(p)$. A point $p$ is a *boundary point* of a set $S$ iff every neighbourhood of $p$ intersects both $S$ and $\neg S$. The *boundary* $\partial S$ of a set $S$ is the set of boundary points of $S$. The *closure* of a set $S$, denoted by $Cl(S)$, is the set $S \bigcup \partial S$. The *interior* of a set, denoted $Int(S)$, is the set $S - \partial S$. If $S$ is open, then $S = Int(S)$; and if $S$ is closed then $S = Cl(S)$.

A *homeomorphism* between topological spaces $X$ and $Y$ is a continuous bijection between $X$ and $Y$. A topological space is a *Hausdorff space* iff for each pair of distinct points $p, q$, there exist disjoint neighbourhoods $O_p$ and $O_q$ of $p$ and $q$ respectively. An *n-manifold* is a Hausdorff space each of whose points has a neighbourhood homeomorphic to an open ball in $\mathbf{R}^n$.

A set $S$ is *convex* iff for any points $a,b \in S$, the set $\gamma a + (1-\gamma)b$, for $0 \leq \gamma \leq 1$, is a subset of $S$. A set $S$ is *connected* iff for any two points $a,b \in S$, there exists a sequence of points $p_0, p_1, \ldots, p_n$ with $a = p_0$ and $b = p_n$ such that for each $i$, $0 \leq i < n$, there is a subset of $S$ homeomorphic to a convex set that contains both $p_i$ and $p_{i+1}$. A set $S$ is *bounded* in $\mathbf{R}^n$ iff there exists a ball of finite radius, containing $S$.

An *n-cell* is a closed subspace of a topological space $T$ with interior homeomorphic to $\mathbf{R}^n$ and whose boundary is non-null. A *closed cell complex* of $T$ is a finite collection $C$ of cells such that:

(1) the interiors of the cells of $C$ are pairwise disjoint;

(2) for each cell $c \in C$ the boundary $\partial c$ of $c$ is the union of elements of $C$;

(3) if $c, d \in C$ and $c \cap d \neq \phi$, then $c \cap d$ is the union of elements of $C$.

An $n$-simplex is a subset of $\mathbf{R}^n$ that is a linear combination of $n + 1$ independent points. A cell complex $C$ is a *simplicial complex* iff each cell $c$ of $C$ is a simplex.

# Appendix B. A Star-Edge Data Structure

In order to claim the time bounds for the algorithms given in "Chapter 4. Intersecting Solids", we must show how the Star-Edge representation is encoded in the actual implementations.

A solid is represented by a list of shells, and each shell $R$ is presented by lists of its vertices, edges and faces:

$$R = [\,Vertices(R), Edges(R), Faces(R)\,].$$

An Edge $e$ is represented as a triple. The first two elements are pointers to the bounding vertices of the edge, denoted $Vertex_1(e)$ and $Vertex_2(e)$. The third element is a list of directed edges, sorted by face direction around $e$ on the plane with normal $Vertex_2(e) - Vertex_1(e)$:

$$[\,Vertex_1(e),\ Vertex_2(e),\ DirectedEdgesAtEdge(e)\,].$$

A face $f$ is a represented as a pair. The first element in the pair is a list of isolated vertices and bounding directed edge cycles that form the "boundary" of $f$. The second element is the equation of the oriented plane of $f$:

$$[\,Boundary(f),\ EquationOnFace(f)\,].$$

A vertex $v$ is a represented as a pair. The first element contains the actual coordinates of the vertex, and the second element is a list, for each face $f$ adjacent to $v$, of directed edges adjacent to $v$ on $f$. Each list of directed edges is sorted radially-clockwise around $v$ on $f$:

$$\begin{bmatrix} PointAtVertex(v), \\ \quad (f_1,\ DirectedEdgesAtVertexOnFace(v, f_1)), \\ \quad (f_2,\ DirectedEdgesAtVertexOnFace(v, f_2)), \\ \quad \bullet\ \bullet\ \bullet \\ \quad (f_k,\ DirectedEdgesAtVertexOnFace(v, f_k)) \end{bmatrix}.$$

A directed edge $\vec{e}$ is represented as a quadruple containing a pointer to the associated edge, an orientation bit, a pointer to the containing cycle and a pointer to the successor directed edge around the containing cycle:

$$\begin{bmatrix} \mathit{EdgeOfDirectedEdge}(\vec{e}), \\ \mathit{RightOrientation?}(\vec{e}), \\ \mathit{CycleOfDirectedEdge}(\vec{e}), \\ \mathit{NextDirectedEdgeAroundCycle}(\vec{e}) \end{bmatrix}.$$

A bounding directed edge cycle $c$ is encoded as a pair. The first element is a pointer to the face that this cycle bounds, and the second is a pointer to a distinguished arbitrary directed edge on this cycle:

$$[\mathit{FaceOfCycle}(c), \mathit{DirectedEdgeOnCycle}(c)].$$

An isolated vertex $I$ is encoded as a pair. The first element is a pointer to the containing face, and the second is a pointer to the vertex itself:

$$[\mathit{FaceOfIsolatedVertex}(I), \mathit{VertexOfIsolatedVertex}(I)].$$

In order for the time bounds of the algorithms in "Chapter 4. Intersecting Solids" to be realized, the actual data structures for vertices, faces, edges, and isolated vertices are augmented with an extra "scratch" field used by the intersection algorithm for bookkeeping.

The relationships described here among the components of the Star-Edge data structure are summarized in Figure 53 on page 108.

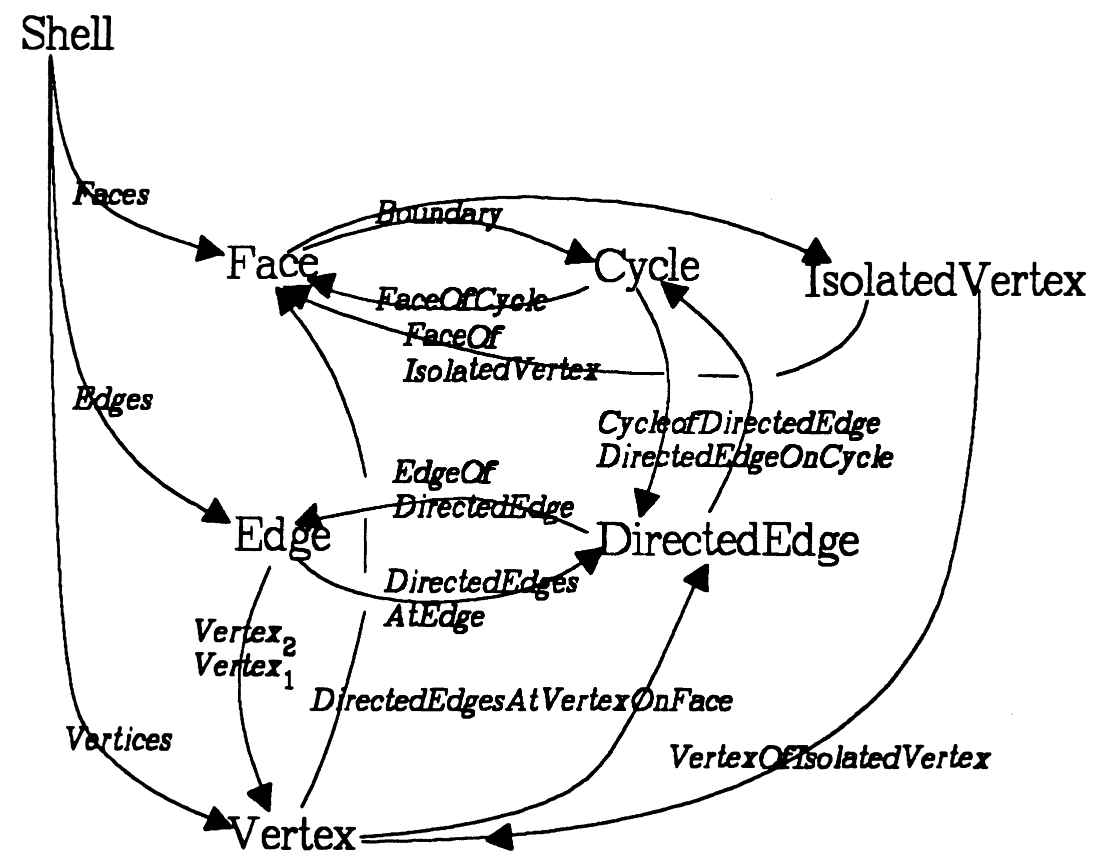



**Figure 53. The Star-Edge Data Structure**

# Appendix C. Geometric Primitives

In order for the proof of correctness of "Chapter 4. Intersecting Solids" to be valid, it must be possible to implement the numerical procedures of "Chapter 4. Intersecting Solids" and "Chapter 5. Incidence Tests" using some form of exact arithmetic. In particular, we must be able to:

(1) determine if a point is below, above, or on an oriented plane;
(2) radially sort faces around an incident edge; and
(3) radially sort coplanar directed edges around an incident vertex.

We now discuss how to implement these tests using rational arithmetic.

Recall from "Chapter 5. Incidence Tests" that the point/plane incidence test is implemented as follows: given plane $P(v) = N \bullet v + d$, a point $a$ is on $P$ if:

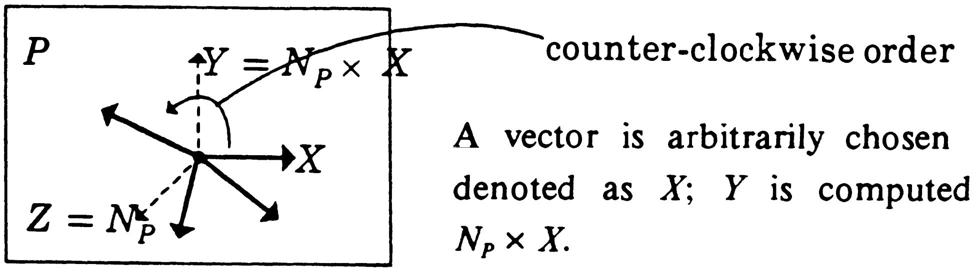
$$\frac{|N \bullet a + d|}{\|N\|} \leq \varepsilon.$$

We can evaluate this expression using rational arithmetic by rewriting it as

$$(N \bullet a + d)^2 \leq \varepsilon^2 (N \bullet N),$$

and if $\varepsilon = 0$, then point $a$ is on $P$ if $N \bullet a + d = 0$.

In order to radially sort faces around an incident edge or radially sort coplanar directed edges around an incident vertex it is necessary to sort coplanar vectors radially around a point on the plane. The implementation of the regularized intersection algorithm uses a simple technique which radially orders a set of coplanar unit vectors relative to a local cartesian coordinate system on the common plane ($P$). First, arbitrarily choose one of the unit vectors and denote it as $X$. Then compute $Y = N_P \times X$, where $N_P$ is the normal vector to $P$ (see Figure 54 on page 110). Using this coordinate system, two unit vectors on $P$ are radially ordered using Algorithm 34 on page 110.




A vector is arbitrarily chosen and denoted as $X$; $Y$ is computed as $N_P \times X$.

**Figure 54. Embedding a cartesian coordinate system on a plane**

Given unit vectors $X$ and $Y$ that define a cartesian coordinate system on a plane $P$, and given unit vectors $\hat{u}$ and $\hat{v}$ in $P$, $\hat{u}$ and $\hat{v}$ are radially ordered (counterclockwise, relative to $X$) by computing a pseudo-angle between $X$ and each of $\hat{u}$ and $\hat{v}$.

Given a unit vector $\hat{w}$ on the plane $P$ spanned by $X$ and $Y$, the pseudo-angle $\alpha$ between $X$ and $\hat{w}$ is computed using the following procedure:

(1) if $\hat{w} \bullet Y \geq 0$ then $\alpha \leftarrow -1 - \hat{w} \bullet X$; otherwise

(2) $\hat{w} \bullet Y < 0$ and $\alpha \leftarrow 1 + \hat{w} \bullet X$.

The unit vectors $\hat{u}$ and $\hat{v}$ are radially ordered by comparing their computed pseudo-angles.

**Algorithm 34. Radially ordering coplanar unit vectors**

The unit vectors of Algorithm 34 can have irrational components, and in order to validate the proof of "Chapter 4. Intersecting Solids", it must be possible to radially order a set of rational vectors using exact arithmetic. As before, arbitrarily choose one of the vectors and denote it as $X$. Then compute $Y = N_P \times X$, where $N_P$ is the normal vector to $P$. Two vectors $u$ and $v$ are radially ordered by first localizing them to quadrants or principal axes. Then if $u$ and $v$ are in the same quadrant, they are radially ordered using $u \bullet X$ and $v \bullet X$. We define

$$\cos\alpha = \frac{u \bullet X}{\|u\|\|X\|} \text{ and } \cos\beta = \frac{v \bullet X}{\|v\|\|X\|}.$$

We observe that if $u$ and $v$ are in the same quadrant, then $\cos\alpha$ and $\cos\beta$ have the same sign, and we can radially order $u$ and $v$ by comparing $\cos^2\alpha$ with $\cos^2\beta$. In the first or third quadrants, $u$ is ordered before $v$ if $\cos^2\alpha < \cos^2\beta$:

$$\cos^2\alpha \; < \; \cos^2\beta \;\text{ iff}\; \frac{(u \bullet X)^2}{u \bullet u} \; > \; \frac{(v \bullet X)^2}{v \bullet v};$$

and in quadrants two and four,

$$\cos^2\alpha \; < \; \cos^2\beta \;\text{ iff}\; \frac{(u \bullet X)^2}{u \bullet u} \; < \; \frac{(v \bullet X)^2}{v \bullet v}.$$

The radial ordering of two rational vectors is given as Algorithm 35.

Given rational vectors $X$ and $Y$ that define a cartesian coordinate system on a plane $P$, and given rational vectors $u$ and $v$ in $P$, $u$ and $v$ are radially ordered (counterclockwise, relative to $X$) using the following procedure:

(1) Localize each of $u$ and $v$ to a quadrant or axis by computing dot products with $X$ and $Y$ (see Figure 55).

(2) If Step (1) does not radially order $u$ and $v$, then $u$ and $v$ are in the same quadrant. If $u$ and $v$ are in the first or third quadrants, then $u$ is ordered before $v$ if

$$(v \bullet v)(u \bullet X)^2 > (u \bullet u)(v \bullet X)^2,$$

and if $u$ and $v$ are in the second or fourth quadrants, then $u$ is ordered before $v$ if

$$(v \bullet v)(u \bullet X)^2 < (u \bullet u)(v \bullet X)^2.$$

**Algorithm 35. Radially ordering coplanar vectors using rational arithmetic**

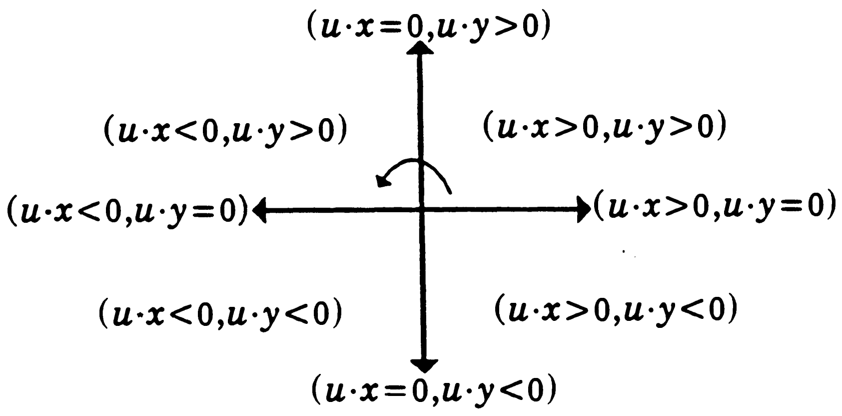



**Figure 55. Localizing a vector to a principal axis or quadrant**

# Appendix D. Benchmarks

The correctness proof of the intersection algorithm, as given in "Chapter 4. Intersecting Solids", assumes exact arithmetic. However, the actual algorithm is implemented using finite-precision arithmetic and the incidence tests of "Chapter 5. Incidence Tests". Experiments that demonstrate the utility of the incidence tests are now described.

The algorithm is implemented in LISP, on a Symbolics LISP machine running the Genera 7.1 operating system. All experimentation is done using finite-precision arithmetic with 53 bits of precision and numerical quantities are tested for equality using a test for absolute error: $a = b$ if $|a - b| \leq \varepsilon$. Unless otherwise stated, all computations are performed using $\varepsilon = 10^{-4}$. For each experiment performed, data for the test solids, where relevant, are presented: Vertex coordinates are given, and for each face, a list of clockwise bounding vertex-cycles is given.

## D.1 *The Compound of Five Cubes*

A first test of the intersection algorithm is the construction of the "compound of five cubes." This solid is obtained by computing the union of the five cubes given in Figure 56 on page 113. Pictorially, the compound is shown in Figure 57 on page 114.

The Star-Edge representation of the compound has 182 vertices, 540 edges, and 30 faces. Each face has 12 triangular bounding directed-edge-cycles (see Figure 58 on page 114). The compound of five cubes is one of the early test objects constructed by the intersection algorithm and is the first non-trivial intersection computed by the algorithm.

**Cube 1**

1 (1.0 −1.0 1.0)
2 (1.0 1.0 1.0)
3 (−1.0 1.0 1.0)
4 (−1.0 −1.0 1.0)
5 (1.0 −1.0 −1.0)
6 (1.0 1.0 −1.0)
7 (−1.0 1.0 −1.0)
8 (−1.0 −1.0 −1.0)

**Cube 2**

1 (1.61803398875 −0.61803398875 0.0)
2 (1.0 1.0 1.0)
3 (−0.61803398875 0.0 1.61803398875)
4 (0.0 −1.61803398875 0.61803398875)
5 (0.61803398875 0.0 −1.61803398875)
6 (0.0 1.61803398875 −0.61803398875)
7 (−1.61803398875 0.61803398875 0.0)
8 (−1.0 −1.0 −1.0)

**Cube 3**

1 (−0.61803398875 0.0 1.61803398875)
2 (0.0 1.61803398875 0.61803398875)
3 (−1.0 1.0 −1.0)
4 (−1.61803398875 −0.61803398875 0.0)
5 (1.0 −1.0 1.0)
6 (1.61803398875 0.61803398875 0.0)
7 (0.61803398875 0.0 −1.61803398875)
8 (0.0 −1.61803398875 −0.61803398875)

**Cube 4**

1 (0.61803398875 0.0 1.61803398875)
2 (1.61803398875 0.61803398875 0.0)
3 (0.0 1.61803398875 −0.61803398875)
4 (−1.0 1.0 1.0)
5 (0.0 −1.61803398875 0.61803398875)
6 (1.0 −1.0 −1.0)
7 (−0.61803398875 0.0 −1.61803398875)
8 (−1.61803398875 −0.618033988750.0)

**Cube 5**

1 (0.61803398875 0.0 1.61803398875)
2 (0.0 1.61803398875 0.61803398875)
3 (−1.61803398875 0.61803398875 0.0)
4 (−1.0 −1.0 1.0)
5 (1.61803398875 −0.61803398875 0.0)
6 (1.0 1.0 −1.0)
7 (−0.61803398875 0.0 −1.61803398875)
8 (0.0 −1.61803398875 −0.61803398875)

**Bounding Face Cycles**

1 (1 2 6 5)
2 (2 3 7 6)
3 (3 4 8 7)
4 (1 5 8 4)
5 (1 4 3 2)
6 (5 6 7 8)

**Figure 56. Data for the compound of five cubes:** Each vertex of each of the five cubes is numbered, and bounding face cycles are given as clockwise vertex-cycles. For example, face (1) on each cube has a single bounding vertex-cycle: vertices (1), (2), (6), and (5).

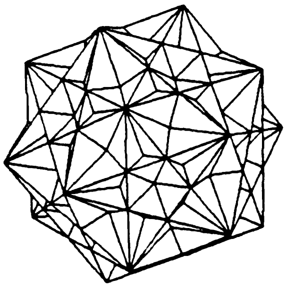
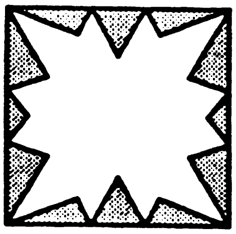


Figure 57. The compound of five cubes



Each face has 12 triangular bounding cycles

Figure 58. A face of the compound of five cubes

The intersection algorithm can be used to construct the solid of Figure 57 in two different ways, and timing results for each are given in Table 1 on page 115:

(i) each union operation is implemented using DeMorgan's laws, and the expression

$$(Cube_1 \bigcup^* Cube_2) \bigcup^* (Cube_3 \bigcup^* Cube_4) \bigcup^* Cube_5$$

is evaluated, entailing five intersection operations, and 15 regularized complementation operations (see "Chapter 3. Some Representational Results"); and

(ii) the complementation operations are discounted, *i.e.*, if $Cube_{\bar{i}}$ is the regularized complement of the $i^{th}$ cube, then the expression

$$(Cube_1^- \bigcap^* Cube_2^-) \bigcap^* (Cube_3^- \bigcap^* Cube_4^-) \bigcap^* Cube_5^-$$

is evaluated.

| Operation | Time (sec) |
|---|---|
| $A \leftarrow Cube_1 \bigcup^* Cube_2$ | 3.2 |
| $B \leftarrow Cube_3 \bigcup^* Cube_4$ | 6.5 |
| $C \leftarrow A \bigcup^* B$ | 37.0 |
| $D \leftarrow C \bigcup^* Cube_5$ | 84.0 |

| Operation | Time (sec) |
|---|---|
| $A \leftarrow Cube_1^- \bigcap^* Cube_2^-$ | 3.3 |
| $B \leftarrow Cube_3^- \bigcap^* Cube_4^-$ | 7.4 |
| $C \leftarrow A \bigcap^* B$ | 35.0 |
| $D \leftarrow C \bigcap^* Cube_5^-$ | 75.5 |

**Table 1. Timing data for the compound of five cubes:** The computation time for the four regularized set operations which, given Star-Edge representations of the five cubes, construct a Star-Edge representation of the compound of five cubes, are given above. The timings include some error due to garbage collection, and so there is some variability. In particular, compare the computation times for the construction of solids $A$ and $B$ using the two different methods.

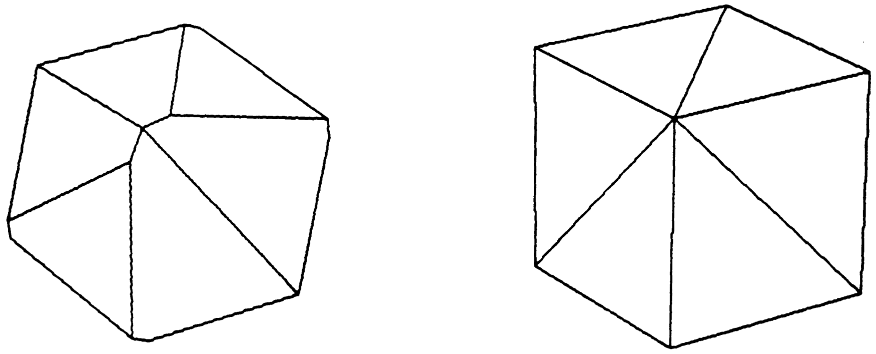
## D.2 Perturbation Experiments

Several experiments done to aid in the development of the incidence tests of "Chapter 5. Incidence Tests" are of the form:

> "Given a Star-Edge representation of solid $A$, construct a Star-Edge representation of solid $B$ by rotating the faces, edges, and vertices of the Star-Edge representation of $A$ about the origin; then compute the intersection of $A$ with $B$."

If the angle of rotation is very small (in comparison with the tolerance $\varepsilon$ of "Chapter 5. Incidence Tests") then any intersection algorithm should announce that $A$ are $B$ are identical. In contrast, if the angle of rotation is sufficiently large then any intersection algorithm should be able to construct a boundary representation of $A \bigcap^* B$ from boundary representations of $A$ and $B$. For example, suppose the perturbation is a small rotation $\theta$ about each of the principal axes: For sufficiently small values of theta, $\theta \leq \alpha$, $A$ and $B$ cannot be distinguished; and for sufficiently large values of theta, $\beta \leq \theta$, a boundary representation of $A \bigcap^* B$ is constructed by an intersection algorithm. However, for rotations of $\alpha < \theta < \beta$, finite-precision intersection algorithms can fail because of mistakes made by the incidence tests, as described in "Chapter 5. Incidence Tests". Using these incidence tests, the *region of instability* $(\beta - \alpha)$ is extremely small for the experiments done, which indicates that symbolic inference techniques can be used to implement geometric tests more robustly.

The first perturbation experiment is the intersection of two nearly identical unit-cubes: Given a Star-Edge representation of a unit cube ($A$) in standard position centred at the origin, a Star-Edge representation of a second unit cube ($B$) is constructed by rotating the faces, edges, and vertices of the Star-Edge representation of $A$ a small amount $\theta$ in turn about each principal axis. Unit cube $A$ is then intersected with unit-cube $B$ (see Figure 59). The unit-cube intersection experiment has been tried on a variety of intersection algorithms that use finite-precision arithmetic, and all of them have a region of instability, *i.e.*, for $\theta$ sufficiently small, each of the algorithms catastrophically fails and is unable to construct a boundary representation of the result.[22]



(a) $\theta = 20$ degrees (b) $\theta = \frac{1}{200}$ degrees

**Figure 59. Intersecting two unit cubes**

Figure 60 on page 117 describes a situation that arises in the intersection of Figure 59(b). Consider an edge of unit-cube $A$ defined by planes $P$ and $Q$, and an edge $e$ of unit-cube $B$ with endpoints $u$ and $v$. If $v$ is in $P$ but not in $Q$, and $u$ is not in $P$, then $e$ cannot intersect the line $P \cap Q$. It is possible to perturb $e$ so that $e$ appears to intersect $P \cap Q$ even though $u$ does not lie in $P$. Any algorithm

[22] The size of the region of instability is dependent on parameters such as the tolerance used in equality tests, the number of bits of precision used in computations, etc. The algorithm of Paoluzzi, Ramella, and Santarelli [1986] fails at $\theta = 1$ degree, and the algorithm of Laidlaw, Trumbore, and Hughes [1986] fails at $0.1 < \theta < 0.5$ degree. The algorithm of Segal and Séquin [1988] fails for $0.00001 < \theta < 0.05$ degree. Several commercial intersection algorithms also fail on this example, but their performance is proprietary and thus cannot be reported here.

which declares that: (i) $e$ intersects $P \bigcap Q$, (ii) $v$ lies on $P$, and (iii) $u$ does not lie on $P$, will fail because it impossible to construct a valid topology for the surface of $A \bigcap^* B$ consistent with these assumptions. In contrast, the incidence tests of "Chapter 5. Incidence Tests" cannot make this mistake because hypotheses (ii) and (iii) are used to show that hypothesis (i) is inconsistent.

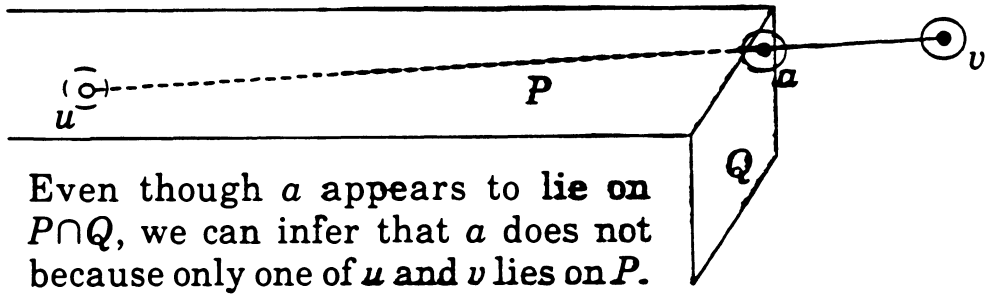



**Figure 60. Topological consistency**

The intersection algorithm of this thesis does fail on the unit-cube example; only by using the bisection technique of Algorithm 36 on page 118 to search for the region of instability is the region found. Using a tolerance of $\varepsilon = 10^{-4}$, the region of instability is $2.8610 \times 10^{-3} < \theta < 2.8686 \times 10^{-3}$ degrees; at $\theta = 2.8610 \times 10^{-3}$ degrees, the intersection algorithm announces that the two cubes are identical; at $\theta = 2.8686 \times 10^{-3}$ degrees, the intersection algorithm produces a duodecahedron; and at the next $\theta$ value examined ($\theta = 2.8648 \times 10^{-3}$ degrees) the intersection algorithm fails. Thus the region of instability is bounded by $\delta\theta = 7.6 \times 10^{-6}$ degrees.

The reason that the intersection algorithm fails on the unit-cube example is easily described. One of the early stages in the algorithm is the computation of $G_P$, embedded planar graph that describes the intersection of an oriented plane $P$ of solid $A$ with solid $B$ (see "Chapter 4. Intersecting Solids"). Because the rotation $\theta$ is very small, there is a face $g$ of $B$ whose vertices are almost incident with $P$. Using the vertex/plane incidence test of "Chapter 5. Incidence Tests", three of the four vertices of $g$ are deemed incident with $P$, and the fourth vertex is deemed below $P$ (see Figure 61 on page 118). Thus $g$ is neither coincident with $P$, nor is there a line of intersection between $g$ and $P$. This failure of the intersection algorithm is not due to any numerical error, but rather illustrates a problem in intersecting nearly coincident faces: in cases such as this, no attempt is made by the incidence tests to infer a line of intersection of $g$ with $P$.

Given a Star-Edge representation of solid $A$, this algorithm iteratively finds a solid $B$ which, when intersected with $A$, causes the intersection algorithm to fail. If no such solid exists, then the algorithm terminates when the region of instability is smaller than the granularity of the finite-precision approximation used to represent the endpoints of the regions.

(1) $\alpha \leftarrow 0.0$ degrees.
$\beta \leftarrow 1.0$ degree.
Repeat Steps (2-3) until either $\alpha = \beta$ (to finite precision) or the intersection algorithm fails.

(2) $\theta \leftarrow \frac{\alpha + \beta}{2}$.
Rotate the faces, edges, and vertices of the Star-Edge representation of $A$ $\theta$ degrees in turn around the $Z$, $Y$, and $X$ axes, forming the Star-Edge representation of $B$.

(3) Intersect $A$ with $B$, yielding $C$. By examining the vertices of $C$, determine if every vertex of $A$ is deemed equal to its rotated counterpart of $B$, and:

(i) if $A$ is deemed identical to $B$ then set $\alpha \leftarrow \theta$; otherwise
(ii) set $\beta \leftarrow \theta$.

**Algorithm 36. Inducing failure in an intersection algorithm**

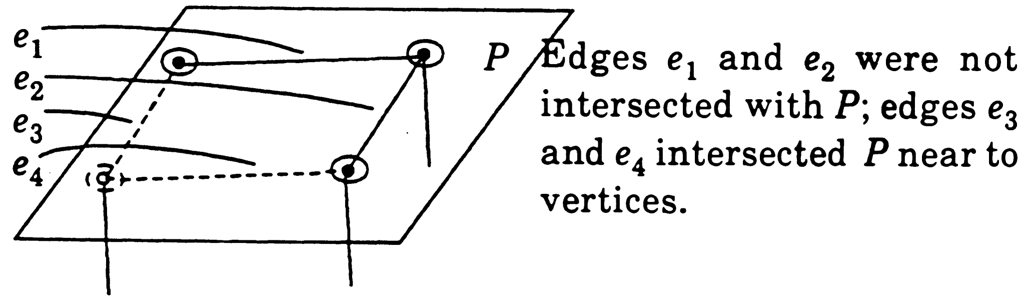



**Figure 61. Failure of the unit-cube experiment**

Results for the unit-cube experiment using values of $\varepsilon$ other than $10^{-4}$ are now presented, demonstrating that to a certain extent, the performance of the intersection algorithm is scalable:

(1) With $\varepsilon = 10^{-2}$, the intersection algorithm fails at $0.2851 < \theta < 0.2891$ degrees, as illustrated in Figure 61, yielding a region of instability of size at most $\delta\theta = 0.0039$ degrees.

(2) With $\varepsilon = 10^{-6}$, the intersection algorithm fails at $2.864783 \times 10^{-5} < \theta < 2.864794 \times 10^{-5}$ degrees as shown in Figure 61, yielding a region of instability of size $O(10^{-10})$ degrees, *i.e.*, one ten-billionth of a degree.

(3) At a tolerance of $\varepsilon = 10^{-8}$, the intersection fails at $2.38 \times 10^{-7} < \theta < 3.57 \times 10^{-7}$ degrees because of a mistake made caused by a numerical error while computing the incidence of a vertex with a plane.

Essentially, the unit-cube benchmark causes the intersection algorithm to fail because the vertices of the square faces of the cubes are not in general position. Just as the incidence tests of "Chapter 5. Incidence Tests" do not reason about vertices defined by more than three planes (and edges defined by more than two planes), neither do the incidence tests reason about faces defined by more than three vertices. Specifically, no finite-precision incidence tests are defined in this thesis that robustly intersect a face with a plane. Thus a reasonable perturbation experiment to try applies Algorithm 36 on page 118 to an equilateral tetrahedron (data for the tetrahedron are given in Figure 62).

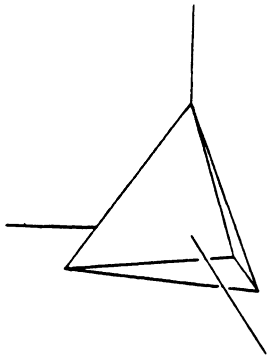


**Figure 62. Constructed tetrahedron:** Vertices and faces of the "unit"-tetrahedron are:

| | Vertices | | Faces |
|---|---|---|---|
| 1 | (−1.0 −0.5773502691896257 −0.408248290463863) | 1 | (4 3 2) |
| 2 | (1.0 −0.5773502691896257 −0.408248290463863) | 2 | (4 1 3) |
| 3 | (0.0 1.1547005383792515 −0.408248290463863) | 3 | (4 2 1) |
| 4 | (0.0 $1.1102230246251565 \times 10^{-16}$ 1.224744871391589) | 4 | (2 3 1) |

Algorithm 36 on page 118 does not cause the intersection algorithm *does not* fail for the solids of Figure 62:

(1) With $\varepsilon = 10^{-2}$, the bisection algorithm does not find a region of instability. At $\theta = 0.2889261375847983$, the intersection algorithm constructs a

pentahedron, and at $\theta = 0.28892613758479824$, the intersection algorithm returns one of the original tetrahedra.

(2) With $\varepsilon = 10^{-4}$, the bisection algorithm does not find a regions of instability. At $\theta = 2.9141868296453733 \times 10^{-5}$ the intersection algorithm constructs a pentahedron, and at $\theta = 2.914186829645373 \times 10^{-5}$ one of the original tetrahedra is returned.

(3) With $\varepsilon = 10^{-6}$, the bisection algorithm does not find a region of instability. At $\theta = 2.91444002899654 \times 10^{-5}$ the intersection algorithm constructs a pentahedron, and at $\theta = 2.9144400289965397 \times 10^{-5}$ one of the original tetrahedra is returned.

The results of the tetrahedron benchmark show that if the region of instability ($\delta\theta$) for the tetrahedral intersection exists, then $\delta\theta$ is smaller than $2^{-53}$.

A different tetrahedron intersection-experiment is obtained by first constructing a boundary representation of the tetrahedron ($A$) of Figure 62 on page 119, and then by constructing a boundary representation of a second tetrahedron ($B$) by independently perturbing each of the vertices of $A$. One would expect an intersection algorithm to fail while trying to intersect $A$ with $B$ if the perturbation is sufficiently small. (This experiment was suggested by Allan Back). The experiment uses the new intersection algorithm to intersect 100 pairs of tetrahedra $A$ and $B$, where the vertices of $B$ are obtained by randomly perturbing the components of the vertex-coordinates of $A$. In all cases the algorithm succeeded in constructing a Star-Edge representation of $A \cap^* B$ (see Table 2).

| | **Number of Faces in $A \cap^* B$** | | | | |
|---|---|---|---|---|---|
| | **4** | **5** | **6** | **7** | **8** |
| **Perturbation ($\tau$):** | **(Frequency)** | | | | |
| $-2\varepsilon \le \tau \le 2\varepsilon$ | 1 | 10 | 31 | 40 | 18 |
| $-\varepsilon \le \tau \le \varepsilon$ | 5 | 29 | 42 | 18 | 6 |

**Table 2. Results of random-tetrahedron intersection-experiment:** Two sets of 100 trials are described. This table describes data like: "for the first experiment, in which individual components of the coordinates of the vertices of $A$ are randomly perturbed by $-2\varepsilon \le \tau \le 2\varepsilon$, 31 of the 100 intersections yield solids with six faces."

The unit-cube benchmark is perhaps one of the easist tests that one might try on an intersection algorithm because the two cubes are symmetric about the origin, and all vertices are orthohedral. Similarly, the tetrahedron benchmarks are easy tests because and the vertices of a tetrahedron are in general position. A more difficult example is given by applying Algorithm 36 on page 118 to the irregular heptahedron of Figure 63: None of the vertices are orthohedral, the object is not symmetric, the vertices of the faces are not in general position, and the centre of rotation used by Algorithm 36 is not contained in the interior of the heptahedron.

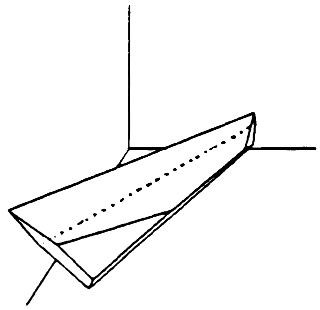


**Figure 63. Data used to construct heptahedron:** Vertices and faces of the heptahedron are:

| | **Vertices** | | **Faces** |
|---|---|---|---|
| 1 | (0.0 −1.0 −2.0) | 1 | (10 9 8 7) |
| 2 | (0.6666666666666667 0.3333333333333326 −4.0) | 2 | (10 7 6 5) |
| 3 | (0.3333333333333335 0.6666666666666667 −4.0) | 3 | (4 9 10 5 3) |
| 4 | (1.4000000000000004 1.0 −4.0) | 4 | (8 9 4 2 1) |
| 5 | ( −1.0 0.0 −2.0) | 5 | (6 7 8 1) |
| 6 | ( −1.0 −1.0 −1.0) | 6 | (2 3 5 6 1) |
| 7 | (2.0 0.5000000000000002 −0.2500000000000001) | 7 | (4 3 2) |
| 8 | (2.0 0.3333333333333335 −0.6666666666666672) | | |
| 9 | (2.0 1.0 −2.5) | | |
| 10 | (2.0 0.75 −0.5) | | |

The bisection algorithm (Algorithm 36) *does not* cause the intersection algorithm to fail this benchmark. The results are similar to the results of the rotated tetrahedron intersection.

(1) With $\varepsilon = 10^{-2}$, the bisection algorithm does not find a regions of instability. At $\theta = 0.018739636635571332$, the intersection algorithm constructs a heptahedron that is distinct from the original ones, and at $\theta = 0.018739636635571\,32$, one of the original heptahedra is returned.

(2) With $\varepsilon = 10^{-4}$, the bisection algorithm does not find a region of instability. At $\theta = 1.8683966414379427 \times 10^{-4}$ the intersection algorithm constructs a heptahedron that is distinct from the original ones, and at $\theta = 1.8683966414379425 \times 10^{-4}$ one of the original heptahedra is returned.

(3) With $\varepsilon = 10^{-6}$, the bisection algorithm does not find a region of instability. At $\theta = 1.8683412011679224 \times 10^{-6}$ the intersection algorithm constructs a heptahedron that is distinct from the original ones is generated, and at $\theta = 1.8683412011679222 \times 10^{-6}$ one of the original heptahedra is returned.

The perturbation experiments presented thusfar intersect solids with valence-three vertices. Recall from "Chapter 5. Incidence Tests" that no claims are made about the robustness of the implementation for vertices of valence higher than three. To demonstrate this, Algorithm 36 is applied to a symmetric octahedron centred at the origin (see Figure 64), and the region of instability is found.

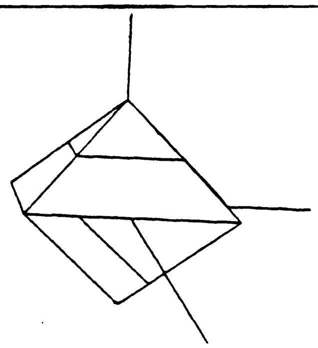


**Figure 64. Intersected octahedra ($\varepsilon$ = $10^{-6}$; $\theta$ = 0.001 degrees):** Vertices and faces of the "unit"-octahedron are:

| Vertices | | Faces | |
|---|---|---|---|
| 1 | (0 0 1) | 1 | (1 3 4) |
| 2 | (0 0 −1) | 2 | (1 6 3) |
| 3 | (1 1 0) | 3 | (1 5 6) |
| 4 | (1 −1 0) | 4 | (1 4 5) |
| 5 | (−1 −1 0) | 5 | (2 3 6) |
| 6 | (−1 1 0) | 6 | (2 4 3) |
| | | 7 | (2 5 4) |
| | | 8 | (2 6 5) |

(1) With $\varepsilon = 10^{-2}$, the bisection algorithm finds a region of instability at $0.20 < \theta < 0.57$ degrees, of size most $\delta\theta = 0.37$ degrees.

(2) With $\varepsilon = 10^{-4}$, the bisection algorithm finds a region of instability at $0.0019 < \theta < 0.0057$ degrees, of size at most $\delta\theta = 0.0038$ degrees.

(3) With $\varepsilon = 10^{-6}$, the bisection algorithm finds a region of instability at $1.9 \times 10^{-5} < \theta < 5.7 \times 10^{-5}$ degrees, of size at most $\delta\theta = 3.8 \times 10^{-5}$ degrees.

Even though no claims are made about the robustness of the intersection algorithm for intersections of solids with high-valence vertices, the region of instability is still relatively small for this example, of size $40\varepsilon$.

## *D.3 Approximating Quadric Surfaces*

The final benchmark presented constructs an approximation to a unit-sphere as follows: beginning with a Star-Edge representation of unit-cube $A_0$ in standard position centred at the origin, construct a Star-Edge representation of $B_0$ by rotating $A_0$ 45 degrees around the $X$-axis, and form a Star-Edge representation of $A_1$ by intersecting $A_0$ with $B_0$. Next, rotate $A_1$ 45 degrees around the $Y$-axis to yield $B_1$, and intersect $A_1$ with $B_1$ to yield $A_2$. Solid $A_3$ is obtained by rotating 45 degrees around the $Z$-axis and intersecting. The procedure terminates when the spherical approximation of $A_{3i}$ has over 1000 faces. The result, shown in Figure 65 on page 124, has the largest Star-Edge representation generated to date, with 3034 faces, 8236 edges, and 5204 vertices.

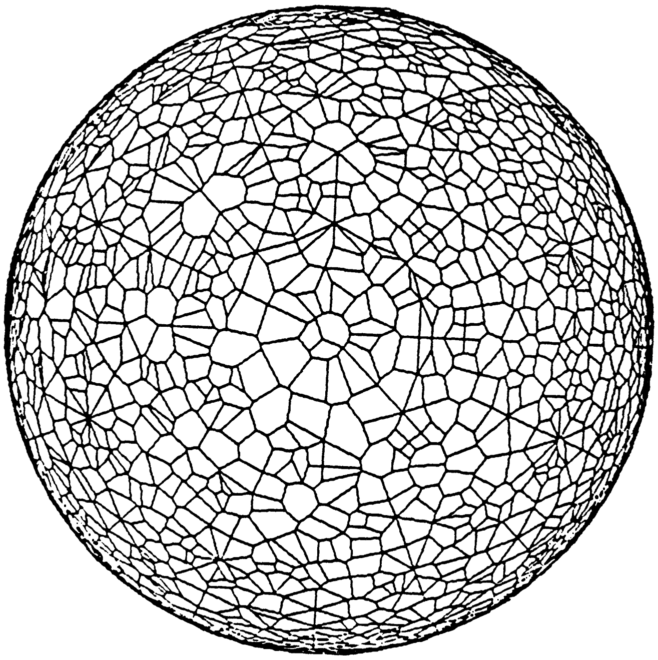


**Figure 65. Polyhedral approximation to a sphere**

The algorithm used to construct the solid of Figure 65 is interesting for several reasons. The representation size (total of the number of vertices, edges, and faces) doubles in each of the twelve iterations, and so the computation time required to perform the final intersection is significant. In fact, the timing data shown in Table 3 on page 125 are the result of many trials; the LISP implementation of the intersection algorithm seems to induce machine failure after computing for several hours[23] and the data of Table 3 on page 125 are the best obtained before a "machine crash." The data of Table 3 also illustrate the utility of a preprocessor that eliminates extraneous computation (see Appendix E). Using this preprocessor, the implemented intersection algorithm, which has asymptotically quadratic performance for the intersections used to construct the solid of Figure 65, intersects two convex polyhedra in linear time.

[23] This example has been tried on several different types of LISP machine and all machines fail because of memory management errors, *i.e.*, the implementation exercises memory management algorithms implemented in the Symbolics operating system.

| Time to generate (sec) | | | Output size | | |
|---|---|---|---|---|---|
| *Solid* | *Preprocessing* | *Total* | *Faces* | *Edges* | *Vertices* |
| $A_0$ | | | 6 | 12 | 8 |
| $A_1$ | 0 | 5 | 10 | 24 | 16 |
| $A_2$ | 0 | 10 | 18 | 36 | 20 |
| $A_3$ | 2 | 21 | 34 | 84 | 52 |
| $A_4$ | 2 | 55 | 58 | 144 | 88 |
| $A_5$ | 5 | 71 | 98 | 226 | 130 |
| $A_6$ | 14 | 128 | 162 | 406 | 246 |
| $A_7$ | 28 | 242 | 266 | 676 | 412 |
| $A_8$ | 63 | 434 | 434 | 1076 | 644 |
| $A_9$ | 129 | 771 | 710 | 1778 | 1070 |
| $A_{10}$ | 268 | 1536 | 1150 | 2988 | 1840 |
| $A_{11}$ | 501 | 3306 | 1866 | 4880 | 3016 |
| $A_{12}$ | 1085 | 6751 | 3034 | 8236 | 5204 |

**Table 3.** **Timing data for sphere approximation:** Each row describes the preprocessing time, total intersection time, and representation output size at each iteration. Note that these data describe total elapsed time, as opposed to computation time.

# Appendix E. Eliminating Extraneous Incidence Tests

The regularized intersection algorithm of "Chapter 4. Intersecting Solids" has been augmented with a preprocessor (written by J. Sasaki) that, given Star-Edge representations of $A$ and $B$, identifies intersecting face-pairs of $A$ and $B$. The preprocessor works as follows: each face is enclosed in an orthohedral box, and the set of boxes is intersected, using a simple variant of an interval tree [Mehlhorn, 1984] 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.

In order to interface this preprocessor with the intersection algorithm, two small changes are made to the intersection algorithm:

(1) In the first stage of the algorithm, plane $P$ of a face $f$ of solid $A$ is intersected with solid $B$, yielding an embedded planar graph called $G_P$. The preprocessor identifies a set of faces of $B$ that might intersect $f$, and only these faces are intersected with $P$. Consequently, it may no longer possible to organize the edges of $G_P$ into bounding cycles. Instead, the edges are organized into chains. Each edge of $G_P$ is either (i) a line segment across a face of $B$ that may intersect face $f$ of $A$, or (ii) an edge of $B$ that may intersect face $f$ of $A$.

(2) In the second stage of the algorithm, it is possible that an edge-chain of $G_P$ does not intersect the boundary of $f$. Before testing to see if $f$ is contained in the interior of $B$ (using the cycle-containment test described in "Chapter 4. Intersecting Solids") it is necessary to augment $G_P$ with the edges and vertices eliminated by the preprocessor.

A good example of the use of the preprocessor is given in Appendix D, where a sphere is approximated using repeated intersections. With the preprocessor, the intersection algorithm requires only linear time to intersect two convex polyhedra. As shown in Table 3 on page 125 the representation sizes, preprocessing time, and total computation time double at each iteration.

In order to understand why the total computation times of Table 3 are empirically linear (as opposed to the asymptotic bound of "Chapter 4. Intersecting Solids") it is necessary to examine the computation steps more carefully. The first preprocessing step boxes every face of solids $A$ and $B$ (requiring examination of every face, edge, and vertex of $A$ and $B$) ; the second preprocessing step sorts the boxes in a principal direction; the third step is a traversal of the tree; and the final step is a search of the tree. For the total preprocessing time to be empirically linear, the constant of computation of the first and third steps must be large relative to the constant of computation of the other steps. Because the preprocessor reports that each face of $A$ may intersect only a constant number of faces of $B$, each construction of a cross-sectional graph $G_P$ requires constant time. Subsequent computations are also done quickly, and the total computation time (each iteration of Table 3) is empirically linear in the size of the input Star-Edge representations.

# References


[1] Aho, A.V., Hopcroft, J.E., Ullman, J.D., *Data Structures and Algorithms*, Addison Wesley, Reading, MA, 1983.

[2] Alt, H., Mehlhorn, K., Wagener, H., Welzl, E., "Congruence, Similarity, and Symmetries of Geometric Objects," *Discrete and Computational Geometry* **3**, 1988, pp. 237-256.

[3] Atkinson, M.D., "An Optimal Algorithm for Geometrical Congruence," *Journal of Algorithms* **8**, 1987, pp. 36-47.

[4] Bentley, J.L., Ottmann, T.A., "Algorithms for Reporting and Counting Intersections," *IEEE Transactions on Computers*, Sept. 1979, pp. 643-647.

[5] Baumgart, B.G., *Winged-Edge Polyhedron Representation*, Rep. CS-320, Stanford A.I. Lab, Stanford Univ., Stanford, CA., 1972.

[6] 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.

[7] Brown, C.M., "Some Mathematical and Representational Aspects of Solid Modeling," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, July 1981, pp. 444-453.

[8] Brown, K.Q., *Fast Intersection of Half-Spaces*, Report CMU-CS-78-129, Dept. of Computer Science, Carnegie Mellon Univ., Pittsburgh, PA, 1978.

[9] Chazelle, B., Edelsbrunner, H., *An Optimal Algorithm for Intersecting Line Segments in the Plane*, Report UIUCDS-R-88-1419, Dept. of Computer Science, Univ. of Illinois., Urbana, IL, 1988.

[10] Dobkin, D.P., Silver, D., "Recipes for Geometry and Numerical Analysis - Part I: An Empirical Study," *Proceedings of the* $4^{th}$ *Symposium on Computational Geometry*, Univ. of Illinois, June 1988, pp. 93-105.

[11] Dobkin, D.P., Laszlo, M.J., "Primitives for the Manipulation of Three-Dimensional Subdivisions," *Proceedings of the* $3^{rd}$ *Symposium on Computational Geometry*, Waterloo, Ont., June 1987, pp. 86-99.

[12] Eastman. C.M., Preiss, K., "A Review of Solid Shape Modelling Based on Integrity Verification," *Computer Aided Design*, March 1984, pp. 66-80.

[13] Edelsbrunner, H., O'Rourke, J., Seidel, R., "Constructing Arrangements of Lines and Hyperplanes with Applications," *SIAM Journal of Computing*, May 1986, pp. 341-363.

[14] Greene, D.H., Yao, F.F., "Finite-Resolution Computational Geometry," *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, Toronto, Ont., Oct. 1986, pp. 143-152.

[15] Grunbaum, B., *Convex Polytopes*, Interscience, London, 1967.

[16] Guibas, L., Stolfi, J., "Primitives for the Manipulation of General Subdivisions and the Computations of Voronoi Diagrams," *ACM Transactions on Graphics*, April 1985, pp. 74-123.

[17] Hanrahan, P.M., *Topological Shape Models*, Phd Thesis, Univ. of Wisconsin, Madison, WI, 1985.

[18] Hertel, S., Mantyla, M., Mehlhorn, K., Nievergelt, J., "Space Sweep Solves Intersection of Convex Polyhedra," *Acta Informatica*, Dec. 1984, pp. 501-519.

[19] Hoffmann, C.M., *The Problem of Accuracy and Robustness in Geometric Computation*, Report CSD-TR-771, Dept. of Computer Science, Purdue Univ., West Lafayette, IN, 1988.

[20] Hoffmann, C.M., Hopcroft, J.E., Karasick, M.S., "Towards Implementing Robust Geometric Computations," *Proceedings of the* $4^{th}$ *Symposium on Computational Geometry*, Univ. of Illinois, June 1988, pp. 106-117.

[21] Karasick, M.S., "The Same Object Problem for Polyhedral Solids," to appear in *Computer Vision, Graphics, and Image Processing*, 1989.

[22] Kirkpatrick, "Optimal Search in Planar Subdivisions," *SIAM Journal of Computing*, Jan. 1983, pp. 28-35.

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

[24] Laidlaw, D.H., Trumbore, W.B., Hughes, J,F., "Constructive Solid Geometry for Polyhedral Objects," Brown Univ., Providence, RI, 1986.

[25] Lee, Y.T., Requicha, A.A.G., "Algorithms for Computing the Volume and Other Integral Properties of Solids, I: Known Methods and Open Issues," *Communications of the ACM*, Sept. 1982, pp. 635-641.

[26] Lee, Y.T., Requicha, A.A.G., "Algorithms for Computing the Volume and Other Integral Properties of Solids, II: A Family of Algorithms Based on Representation Conversion and Cellular Approximation," *Communications of the ACM*, Sept. 1982, pp. 642-650.

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

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

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

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

[31] Milenkovic, V.J., *Verifiable Implementations of Geometric Algorithms Using Finite Precision Arithmetic*, PhD Thesis, Carnegie Mellon Univ., Pittsburgh, PA, 1988.

[32] Muller, D.E., Preparata, F.P., "Finding the Intersection of Two Convex Polyhedra," *Theoretical Computer Science*, Oct. 1978, pp. 217-236.

[33] O'Connor, M.A., Rossignac, J.A., "Robust Utilities for Geometric Modelling with Natural Quadrics," paper presented at the *SIAM Conference on Applied Geometry*, Albany, NY, July 1987.

[34] Ottmann, T., Thiemt, G., Ullrich, C., "Numerical Stability of Geometric Algorithms," *Proceedings of the 3rd Symposium on Computational Geometry*, Waterloo, Ont., June 1987, pp. 119-125.

[35] 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.

[36] Preparata, F.P., Hong, S.J., "Convex Hulls of Finite Points Sets in Two and Three Dimensions," *Communications of the ACM*, Feb. 1977, pp. 87-93.

[37] Preparata, F.P., Muller, D.E., "Finding the Intersection of $n$ Half-Spaces in Time $O(n \log n)$," *Theoretical Computer Science*, Jan. 1979, pp. 45-55.

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

[39] Requicha, A.A.G., *Mathematical Models of Rigid Solid Objects*, Tech. Memo 28, Production Automation Project, Univ. of Rochester, Rochester, NY, 1977.

[40] Requicha, A.A.G., "Representations of Rigid Solids: Theory, Methods, and Systems," *ACM Computing Surveys*, Dec. 1980, pp. 437-464.

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

[42] Requicha, A.A.G., Voelcker, H.B., "Solid Modeling: A Historical Summary and Contemporary Assessment," *IEEE Computer Graphics and Applications*, March 1982, pp. 9-24.

[43] Requicha, A.A.G., Voelcker, H.B., "Boolean Operations in Solid Modeling: Boundary Evaluation and Merging Algorithms," *Proceedings of the IEEE*, Jan. 1985, pp. 30-44.

[44] Rossignac, J.R., Voelcker, H.B., *Active Zones in Constructive Solid Geometry for Redundancy and Interference Detection*, Research Report, IBM T. J. Watson Research Centre, Yorktown Heights, NY, 1986.

[45] Rourke, C.P., Sanderson, B.J., *Introduction to Piecewise-Linear Topology*, Springer Verlag, 1982.

[46] Segal, M., Séquin, C., "Consistent Calculation for Solid Modeling," *Proceedings of the* 1*st* *Symposium on Computational Geometry*, Baltimore, MD, June 1985, pp. 29-38.

[47] Segal, M., Séquin, C., "Partitioning Polyhedral Objects into Nonintersecting Parts," *IEEE Computer Graphics and Applications*, January 1988, pp. 53-67.

[48] Silva, C.E., *Alternative Definitions of Faces in Boundary Representations of Solid Objects*, Tech. Memo 36, Production Automation Project, Univ. of Rochester, Rochester, NY, 1981.

[49] Sinha, P., *Surface-Surface Intersections for Geometric Modeling*, Phd Thesis, Cornell Univ., Ithaca, NY, 1986.

[50] Sugihara, K., "An *n* log *n* Algorithm for Determining the Congruity of Polyhedra," *Journal of Computer and System Sciences*, **29**, 1984, pp. 36-47,

[51] Sugihara, K., "An Approach to Error-Free Solid Modeling," Notes, Institute for Mathematics and its Applications, Univ. of Minnesota, 1987.

[52] Thomas, S.W., *Modelling Volumes Bounded by B-Spline Surfaces*, PhD Thesis, Dept. of Computer Science, Univ. of Utah, Salt lake City, Utah, 1984.

[53] Tilove, R.B., *Exploiting Spatial Locality and Structural Locality in Geometric Modelling*, PhD Thesis, College of Engineering and Applied Science, Univ. of Rochester, Rochester, NY, 1981.

[54] Tilove, R.B., "A Null Object Detection Algorithm for Constructive Solid Geometry," *Communications of the ACM*, July 1984, pp. 684-694.

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

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

# Index

## A


adjacency mechanisms of a representation 19
areas of $G_P$ 51


## B


$B - Rep_{\min}$
- canonical encoding of 34
- edge 18
- face-boundary 18
- face-boundary encoding 33
- face-interior 18
- vertex 18

boolean algebra 29
boundary representation of Laidlaw, Trumbore, and Hughes 27
boundary representation properties 12
- connectivity 15
- constant-neighbourhood property of a face 15
- explicit boundary representation 14
- faces cover the solid's surface 12
- faces have pairwise-disjoint interiors 12

boundary representation ($B - Rep$) 6
- edge of 6
- face of 6
- vertex of 6


## C


closed cell complex 4
constructive solid geometry (CSG) 5
converting B-Reps to the Star-Edge representation 28
cross-section of a solid (see $G_P$)


## D


deemed incidence (see incidence)
degeneracy 7
directed edges 20
- directed edges of an edge 21
- edge of 21
- face direction of 21
- face of 21
- initial vertex of 21
- orientation of 21
- tangent of 21
- terminal vertex of 21

doubly-connected edge list (DCEL) 42
dual 42
- of a convex polyhedron 42
- of a half-space 42


## E


edge (see boundary representation)
encoding of a boundary representation 30

## F

face (see boundary representation)
face-boundary 21
  directed-edge cycles of 22
  inner cycle of 78
  isolated vertices of 22
  outer cycle of 78
  successor in a bounding cycle of 22
Face-Edge representation for cell complexes 26
Facet-Edge representation for cell complexes 26
fundamental assumptions about faces 12

## G

$G_P$
  cross-face edge of 54
  edges and vertices of 51
  representation 51
  unbounded interior of 51

## I

incidence 85
  consistency in testing 86
  deemed incidence 86
  defining in terms of proximity 85
  success of the intersection algorithm 87
  symmetry of 87
  testing for 85
  transitivity of 87

## M

manifold solids 3
minimal boundary representation of a solid (see $B - Rep_{\min}$)
minimum feature size 88

## N

n-cell 4
numerically robust algorithm 9

## O

octtree 5
oriented bounding-plane 15

## P

plane of a face 15
planes of a solid 15

## Q

Quad-Edge boundary representation 24

## R

region of instability 116
regular convex polytope 2
regular set 1
regularized set operations 1
Reg(•) 1

## S

shell of a solid 45
size of a boundary representation 30
solid 2
spatial decomposition 5
Star-Edge Representation 19
  directed edges 20
  edge 19
  face 19
  face-boundary 19
  face-interior 19

size of 30
vertex 19
success of the intersection algorithm (see incidence)


## T


transference of edges and vertices 65


## V


vertex (see boundary representation)


## W


Weiler's technique for representing solids 13
Winged-Edge boundary representation 24
Winged-Triangle boundary representation 24


## 3


3-complex 4